SYNOPSYS®

DesignWare IP Family Quick Reference Guide

To search the entire manual set, press this toolbar button. For help, refer to intro.pdf.

April 2003

Copyright Notice and Proprietary Information

Copyright © 2003 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Calaveras Algorithm, CoCentric, COSSAP, CSim, DelayMill, Design Compiler, DesignPower, DesignWare, Device Model Builder, Enterprise, EPIC, Formality, HSPICE, Hypermodel, I, InSpecs, in-Sync, LEDA, MAST, Meta, Meta-Software, ModelAccess, ModelExpress, ModelTools, PathBlazer, PathMill, Physical Compiler, PowerArc, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SmartLogic, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, TestBench Manager, Testify, TetraMAX, TimeMill, TMA, VERA, and VeriasHDL are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAII, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, Cosmos SE, CosmosLE, Cosmos-Scope, Cyclelink, Davinci, DC Expert, DC Expert *Plus*, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler SoCBIST, Direct RTL, Direct Silicon Access, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FormalVera, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, Frameway, Gatran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, Integrator, Interactive Waveform Viewer, iQBus, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, LRC, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, NanoSim, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLlint, OpenVera, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, Progen, Prospector, Proteus OPC, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saber Co-Simulation, Saber for IC Design, SaberDesigner, SaberGuide, SaberRT, SaberScope, SaberSketch, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-Hspice, Star-HspiceLink, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-Sim XT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-OPC, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, Test Compiler, TestGen, TetraMAX TenX, The Power in Semiconductors, TheHDL, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

DesignSphere, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

Contents

Preface **11**
About This Manual 11
Manual Overview 11
Typographical and Symbol Conventions 12
Synopsys Common Licensing (SCL) 12
Getting Help 13
Additional Information 13
Comments? 13

Chapter 1
Overview **15**
DesignWare Library 16
Building Block IP 17
AMBA On-Chip Bus 17
Microprocessors/Microcontroller Cores 19
Memory IP 20
Verification Models of Bus and I/O Standards 20
Board and FPGA Verification IP 21
DesignWare Verification Library 21
DesignWare Cores 23
DesignWare Star IP 25

Chapter 2
DesignWare Library Synthesizable IP **27**
Building Block IP 27
Building Block IP QuickStart 28
Building Block IP in FPGA Compiler II QuickStart 30
Datapath Generator Overview 32
Datapath – Arithmetic Overview 33
DW01_absval 34
DW01_add 35
DW01_addsub 37
DW_addsub_dx 39
DW01_ash 41
DW_bin2gray 42
DW01_bsh 43
DW01_cmp2 45

DW01_cmp6 46
DW_cmp_dx 48
DW_cntr_gray 50
DW01_csa 51
DW01_dec 52
DW_div 53
DW_div_pipe 55
DW_gray2bin 57
DW01_inc 58
DW01_incdec 59
DW_inc_gray 61
DW02_mac 62
DW_minmax 63
DW02_mult 64
DW02_multp 66
DW02_mult_2_stage 67
DW02_mult_3_stage 68
DW02_mult_4_stage 69
DW02_mult_5_stage 70
DW02_mult_6_stage 71
DW_mult_dx 72
DW_mult_pipe 73
DW02_prod_sum 75
DW02_prod_sum1 77
DW_prod_sum_pipe 79
DW01_satrnd 81
DW_shifter 83
DW_square 85
DW_squarep 86
DW_sqrt 87
DW_sqrt_pipe 88
DW01_sub 90
DW02_sum 92
DW02_tree 93
Datapath – Trigonometric Overview 94
DW02_cos 95
DW02_sin 96
DW02_sincos 97

Datapath – Sequential Overview 98
DW_div_seq 99
DW_mult_seq 101
DW_sqrt_seq 103
Datapath – Floating Point Overview 105
DW_add_fp 106
DW_cmp_fp 107
DW_div_fp 108
DW_flt2i_fp 110
DW_i2flt_fp 111
DW_mult_fp 112
Memory – Registers 113
DW03_pipe_reg 114
DW03_reg_s_pl 115
DW03_shftreg 116
DW04_shad_reg 117
Memory – FIFO Overview 119
DW_asymfifo_s1_df 120
DW_asymfifo_s1_sf 123
DW_asymfifo_s2_sf 127
DW_fifo_s1_df 131
DW_fifo_s1_sf 133
DW_fifo_s2_sf 135
DW_asymfifoctl_s1_df 138
DW_asymfifoctl_s1_sf 141
DW_asymfifoctl_s2_sf 144
DW_fifoctl_s1_df 148
DW_fifoctl_s1_sf 150
DW_fifoctl_s2_sf 152
Memory – Synchronous RAMs 155
DW_ram_r_w_s_dff 156
DW_ram_r_w_s_lat 157
DW_ram_2r_w_s_dff 158
DW_ram_2r_w_s_lat 160
DW_ram_rw_s_dff 161
DW_ram_rw_s_lat 162
Memory – Asynchronous RAMs 163
DW_ram_r_w_a_dff 164
DW_ram_r_w_a_lat 165
DW_ram_2r_w_a_dff 166

DW_ram_2r_w_a_lat 168
DW_ram_rw_a_dff 169
DW_ram_rw_a_lat 170
Memory – Stacks 171
DW_stack 172
DW_stackctl 174
Data Integrity 176
DW_crc_p 177
DW_crc_s 179
DW_ecc 181
DW04_par_gen 183
Data Integrity – Coding Group Overview 184
DW_8b10b_dec 185
DW_8b10b_enc 187
DW_8b10b_unbal 188
Test – JTAG Overview 189
DW_tap 190
DW_tap_uc 192
DW_bc_1 195
DW_bc_2 196
DW_bc_3 197
DW_bc_4 198
DW_bc_5 199
DW_bc_7 200
Logic – Combinational Overview 202
DW01_binenc 203
DW01_decode 204
DW01_mux_any 205
DW01_prienc 206
Logic – Sequential Overview 207
DW03_bictr_dcnto 208
DW03_bictr_scnto 209
DW03_bictr_decode 210
DW_dpll_sd 211
DW03_lfsr_dcnto 213
DW03_lfsr_scnto 214
DW03_lfsr_load 215
DW03_lfsr_updn 216
DW03_updn_ctr 217

Application Specific – Interface Overview 218
DW_debugger 219
Application Specific – Control Logic 221
DW_arbiter_2t 222
DW_arbiter_dp 224
DW_arbiter_fcfs 226
DW_arbiter_sp 228
DSP Library Overview 230
GTECH Library Overview 230
AMBA On-Chip Bus Logic and Peripherals IP 231
DW_ahb 232
DW_ahb_h2h 234
DW_ahb_icm 235
DW_ahb_ictl 236
DW_apb 237
DW_apb_gpio 238
DW_apb_i2c 239
DW_apb_ictl 240
DW_apb_rap 241
DW_apb_rtc 242
DW_apb_ssi 243
DW_apb_timers 245
DW_apb_uart 246
DW_apb_wdt 248
Memory IP 249
DW_memctl 250
DW_rambist 251
Microprocessors/Microcontroller Cores 253
DW8051 254

Chapter 3
DesignWare Library Verification IP 255
Overview 255
DesignWare VMT Models 256
AMBA AHB Models 258
AMBA APB Models 260
Serial Input/Output Interface Models 261

DesignWare FlexModels 262
Listing of FlexModels 262
Ethernet Verification Models 264
PCI / PCI-X Bus Verification Models 265
USB 1.1 / 2.0 Bus Host Interface Model 266
DesignWare Memory Models 267
DesignWare SmartModels 268
SmartModel Features 268
SmartModel Types 268
SmartModel Timing Definitions 269
Specific Model Information 269

Chapter 4
DesignWare Cores 271
dwcore_blueiq 273
dwcore_blueiq_devkit 275
dwcore_ethernet 276
dwcore_ethernet_sub 277
dwcore_gig_ethernet 278
dwcore_gig_ethernet_sub 279
dwcore_pci 280
dwcore_pcix 281
dwcore_pci_express 282
dwcore_usb_otg 283
dwcore_usb1_device 284
dwcore_usb1_host 285
dwcore_usb1_hub 286
dwcore_usb2_host 287
dwcore_usb2_device 288
dwcore_usb2_phy 289
dwcore_1394_avlink 290
dwcore_1394_device 291
dwcore_1394_ohci 292
dwcore_1394_cphy 293
dwcore_jvxtreme 294
dwcore_jpeg_codec 295
dwcore_jpeg2_codec 296
dwcore_jpeg2_encod 297

Chapter 5
DesignWare Star IP 299
DW_IBM440 300
DW_V850E-Star 302
DW_C166S 304
DW_TriCore1 306
DW_MIPS4KE 308
Mezoe Interface Express 310
SMSC USB 2.0 PHY 311

Index 313

Preface

About This Manual

This manual is a brief overview of the DesignWare Library of synthesizable and verification IP. For detailed product information, refer to individual product databooks and manuals mentioned in the following chapters.

Manual Overview

This manual contains the following chapters:

Preface	Describes the manual and typographical conventions and symbols; tells how to get technical assistance.
Chapter 1 “Overview”	Contains an overview and general description of the DesignWare Library product offering.
Chapter 2 “DesignWare Library Synthesizable IP”	Contains a brief description of each DesignWare Library Synthesizable IP.
Chapter 3 “DesignWare Library Verification IP”	Describes the available DesignWare Library verification models.
Chapter 4 “DesignWare Cores”	Contains a brief description of each DesignWare Core.
Chapter 5 “DesignWare Star IP”	Contains a brief description of each DesignWare Star IP core

Typographical and Symbol Conventions

Table 1 lists the conventions that are used throughout this document.

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
Monospace	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or *Italic*	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: `pin1 [pin2 ... pinN]` In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > SubMenu	Pulldown menu paths, such as: **File > Save As …**

Synopsys Common Licensing (SCL)

You can find general SCL information on the Web at:

http://www.synopsys.com/keys

Getting Help

If you have a question about using Synopsys products, please consult product documentation that is installed on your network or located at the root level of your Synopsys product CD-ROM (if available). You can also access documentation for DesignWare products on the Web:

- Product documentation for many DesignWare products:

  http://www.synopsys.com/products/designware/docs

- Navigate to individual DesignWare IP component datasheets:

  http://www.synopsys.com/designware

You can also contact a Synopsys Support Center online or by phone:

- http://www.synopsys.com/support/support.html
- United States:
  Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific Time, Mon—Fri.
- Canada:
  Call 1-650-584-4200 from 7 AM to 5:30 PM Pacific Time, Mon—Fri.
- All other countries:
  Find other local support center telephone numbers at the following URL:

  http://www.synopsys.com/support/support_ctr

Additional Information

General information about Synopsys and its products is available at the following URL:

http://www.synopsys.com

Comments?

To report errors or make suggestions, please send e-mail to:

doc@synopsys.com

To report an error that occurs on a specific page, select the entire page (including headers and footers), and copy to the buffer. Then paste the buffer to the body of your e-mail message. This will provide us with information to identify the source of the problem.

1 Overview

The DesignWare family of products provides designers with a comprehensive portfolio of synthesizable and verification IP solutions targeting ASIC, SoC & FPGA designs. DesignWare products offer an unparalleled advantage by providing engineers with thoroughly tested, silicon-proven IP - all from a single provider, ensuring complete interoperability. The DesignWare family includes the following products:

- “DesignWare Library” on page 16 - most commonly used design building blocks such as FIFO/FIFO Controllers, CRC, ECC, 8b10b... as well as comprehensive AMBA® On-Chip Bus synthesizable and verification IP solution and more.
- “DesignWare Verification Library” on page 21 - a subset of the DesignWare Library and contains reusable, pre-verified verification IP of the industry's most popular bus and interface standards such as AMBA, PCI, PCI-X, PCI Express USB 2.0.
- “DesignWare Cores” on page 23 - silicon-proven, digital and analog standards-based connectivity IP such as PCI, PCI-X, PCI Express, USB 2.0 On-the-Go (OTG), USB 2.0 PHY.
- “DesignWare Star IP” on page 25 - high-performance, high-value IP microprocessor cores from leading Star IP providers such as IBM, Infineon Technologies, MIPS Technologies, and NEC.

DesignWare Library

The DesignWare Library provides designers with a comprehensive collection of synthesizable and verification IP. The library contains the following principal ingredients for ASIC, SoC, and FPGA design and verification:

- Building Block IP (Datapath, Data Integrity, Test...)
- AMBA On-Chip Bus Logic, Peripherals & Verification Models
- Complete Memory Portfolio (Memory Models, Memory BIST, Memory Controller)
- Verification Models of the most popular Bus and I/O Standards (PCI, PCI-X,PCI Express, USB 2.0...)
- Popular Processor Cores from Industry-Leading IP Vendors (IBM, Infineon, MIPS, NEC)
- Board and FPGA Verification IP

A single license gives you access to all the IP in the library. For more information on the DesignWare Library, refer to the following:

http://www.synopsys.com/products/designware

or send email at:

designware@synopsys.com

For a detailed search of the available IP, refer to the following:

http://www.synopsys.com/ipdirectory

Building Block IP

The DesignWare Building Block IP is a collection of over 140 technology-independent, high-quality, high-performance IP. Most of these IP elements include multiple implementations to provide a variety of performance and area tradeoff options.

Component groups for the Building Block IP are identified in the following table. For more detail, refer to "Building Block IP" on page 27.

Component Group	Description	Component Type
Datapath	Arithmetic, floating point, trigonometric, and sequential math IP (page 32)	Synthesizable RTL
Memory	Registers, FIFO, synchronous and asynchronous RAM, and stack IP (page 113)	Synthesizable RTL
Data Integrity	Data integrity IP such as CRC, ECC, 8b10b... (page 176)	Synthesizable RTL
Test	JTAG IP such as boundary scan, TAP controller... (page 189)	Synthesizable RTL
Logic	Combinational, sequential, and control IP (page 221)	Synthesizable RTL
Interface	Debugger IP (page 218)	Synthesizable RTL
DSP	Digital FIR filter IP (page 230)	Synthesizable RTL
GTECH	Technology-independent IP library to aid users in developing technology-independent parts (page 230)	Synthesizable RTL

AMBA On-Chip Bus

AMBA is a standard bus architecture system developed by ARM® for rapid development of processor-driven systems. AMBA also allows a number of bus peripherals and resources to be connected in a consistent way. The following Synopsys AMBA components are AMBA 2.0 compliant:

Component Name	AMBA 2.0 Component Description	Component Type
DW_ahb	AHB bus, arbitration, decode, and control logic (page 232)	Synthesizable RTL
DW_ahb_h2h	AHB to AHB Bridge (page 234)	Synthesizable RTL
DW_ahb_icm	AHB Multi-layer Interconnection Matrix (page 235)	Synthesizable RTL

DW_ahb_ictl	AHB Interrupt Controller (page 236)	Synthesizable RTL
DW_apb	APB bus, decode, and bridge (page 237)	Synthesizable RTL
DW_apb_gpio	APB GPIO (page 238)	Synthesizable RTL
DW_apb_i2c	APB I^2C Interface (page 239)	Synthesizable RTL
DW_apb_ictl	APB Interrupt Controller (page 240)	Synthesizable RTL
DW_apb_rap	APB Remap & Pause (page 241)	Synthesizable RTL
DW_apb_rtc	APB Real Time Clock (page 242)	Synthesizable RTL
DW_apb_ssi	APB Synchronous Serial Interface (page 243)	Synthesizable RTL
DW_apb_timers	APB Timer (page 245)	Synthesizable RTL
DW_apb_uart	APB UART (page 246)	Synthesizable RTL
DW_apb_wdt	APB Watch Dog Timer (page 248)	Synthesizable RTL
DW_memctl	Memory Controller (page 250)	Synthesizable RTL
ahb_act_monitor_vmt	AMBA Compliance Test-bench™ (ACT) This component is licensed separately. (page 258)	Verification Model
ahb_bus_vmt	AHB Bus (page 258)	Verification Model
ahb_master_vmt	AHB Master (page 258)	Verification Model
ahb_monitor_vmt	AHB Monitor (page 258)	Verification Model
ahb_slave_vmt	AHB Slave (page 258)	Verification Model
apb_master_vmt	APB Master (page 260)	Verification Model
apb_monitor_vmt	APB Monitor (page 260)	Verification Model
apb_slave_vmt	APB Slave (page 260)	Verification Model

Microprocessors/Microcontroller Cores

Synthesizable RTL of these cores are available through the Star IP Program. To search for specific devices of the processor models, visit http://www.synopsys.com/ipdirectory.

Component Name	Component Description	Component Type
DW_IBM440 (Available Q2'03)	PowerPC 440 Microprocessor Core from IBM (page 300)	Synthesizable RTL[a] Verification Model
DW_V850E-Star	V850E Processor Core from NEC (page 302)	Synthesizable RTL[a] Verification Model
DW_C166S	16-bit Processor from Infineon (page 304)	Synthesizable RTL[a] Verification Model
DW_TriCore1	TriCore1 32-Bit Processor Core from Infineon (page 306)	Synthesizable RTL[a] Verification Model
DW_MIPS4KE	Processor Core Family from MIPS (page 308)	Synthesizable RTL[a] Verification Model
DW8051	8-bit Microcontroller (page 254)	Synthesizable RTL
AMD	Processor Models	Verification Models
ARM	Processor Models	Verification Models
Fujitsu	Processor Models	Verification Models
IBM	Processor Models	Verification Models
IDT	Processor Models	Verification Models
Infineon	Processor Models	Verification Models
MIPS	Processor Models	Verification Models
Motorola	Processor Models	Verification Models
NEC	Processor Models	Verification Models
TI	Processor Models	Verification Models

a. Verification models of these cores are included in the DesignWare Library. Synthesizable RTL of these cores are available through the Star IP Program, which has more information at: http://www.synopsys.com/designware/star_ip.html.

Memory IP

Component Name	Component Description	Component Type
DesignWare Memory Models	DesignWare contains thousands of pre-verified memory models, with over 10,000 devices from more than 25 vendors. (page 267)	Verification Models
DW_memctl	Memory Controller (page 250)	Synthesizable RTL
DW_rambist	DesignWare Memory BIST solution (page 251)	Synthesizable RTL
DW Memory Building Block IP	DesignWare Building Block IP contains many memory-related IP. (page 113)	Synthesizable RTL

To view the complete DesignWare memory portfolio, refer to the following:

http://www.synopsys.com/memorycentral

Verification Models of Bus and I/O Standards

Component Name	Component Description	Component Type
AMBA	See AMBA On-Chip Bus on page 17	Verification Models
pciexpress_vmt	PCI Express	Verification Model
enettx_fx, enetrx_fx, enethub_fx, rmiirs_fx	IEEE 802.3 Transmitter, Receiver, RMII Interface Reconciliation Sublayer (page 264)	Verification Models
pcimaster_fx, pcislave_fx and pcimonitor_fx	PCI/PCI-X - 32-bit and 64-bit 33 Mhz and 66 Mhz PCI Simulation Model and Test Suite (page 265)	Verification Models
usbhost_fz	USB 2.0 Transceiver Macrocell Interface & Universal Serial Bus 2.0 Host (page 266)	Verification Model
ieee1394a_fx	IEEE 1394a_ PHY_LINK	Verification Model
sio_txrx_vmt, sio_monitor_vmt	Serial Input/Output Interface Models (page 261)	Verification Models

Board and FPGA Verification IP

The DesignWare Library contains over 18,500 simulation models for ASIC, SoC, FPGA and Board verification. For a complete search, visit http://www.synopsys.com/ipdirectory.

Component Group	Component Reference
VMT Models	Refer to “DesignWare VMT Models” on page 256
FlexModels	Refer to “DesignWare FlexModels” on page 262
DesignWare Memory Models	Refer to “DesignWare Memory Models” on page 267
SmartModel Library	Refer to “DesignWare SmartModels” on page 268

DesignWare Verification Library

The DesignWare Verification Library, a subset of the DesignWare Library, contains reusable, pre-verified verification IP of the industry's most popular bus and interface standards, Design Views for Star IP processor cores and over 10,000 memory models. The following table identifies the various components that make up this library.

Component Name	Component Description	Component Type
AMBA AHB Models (refer to page 258)		
ahb_bus_vmt	AHB Bus	Verification Model
ahb_master_vmt	AHB Master	Verification Model
ahb_monitor_vmt	AHB Monitor	Verification Model
ahb_slave_vmt	AHB Slave	Verification Model
AMBA APB Models (refer to page 260)		
apb_master_vmt	APB Master	Verification Model
apb_monitor_vmt	APB Monitor	Verification Model
apb_slave_vmt	APB Slave	Verification Model

DesignWare Bus & I/O Standards		
enettx_fx, enetrx_fx, enethub_fx, rmiirs_fx	IEEE 802.3 Transmitter, Receiver, RMII Interface Reconciliation Sublayer (page 264)	Verification Models
pciexpress_vmt	PCI Express	Verification Model
pcimaster_fx, pcislave_fx and pcimonitor_fx	PCI/PCI-X - 32-bit and 64-bit 33 Mhz and 66 Mhz PCI Simulation Model and Test Suite (page 265)	Verification Models
usbhost_fz	USB 2.0 Transceiver Macrocell Interface & Universal Serial Bus 2.0 Host (page 266)	Verification Model
ieee1394a_fx	IEEE 1394a_ PHY_LINK	Verification Model
sio_txrx_vmt, sio_monitor_vmt	Serial Input/Output Interface Models (page 261)	Verification Models
DesignWare Design Views of Star IP Microprocessors		
DW_IBM440 (Available Q2'03)	PowerPC 440 Microprocessor Core from IBM (page 300)	Verification Model
DW_V850E-Star	V850E Processor Core from NEC (page 302)	Verification Model
DW_C166S	16-bit Processor from Infineon (page 304)	Verification Model
DW_TriCore1	TriCore1 32-Bit Processor Core from Infineon (page 306)	Verification Model
DW_MIPS4KE	Processor Core Family from MIPS (page 308)	Verification Model
DesignWare Memory - Access to the full suite of memory IP is made available through DesignWare Memory Central; a memory-focused Web site that lets designers download DesignWare Memory IP and documentation. Visit Memory Central at: http://www.synopsys.com/products/designware/memorycentral		

Also visit the DesignWare Verification Library web page at
http://www.synopsys.com/products/designware/dwverificationlibrary.html

DesignWare Cores

The DesignWare Cores shown in the following table provide system designers with silicon-proven, digital and analog connectivity IP. DesignWare Cores are licensed individually, on a fee-per-project business model.

Component Name	Component Description	Component Type
Bluetooth Core		
dwcore_blueiq	Bluetooth Baseband and Link Manager (page 273)	Synthesizable RTL
dwcore_blueiq_devkit	Bluetooth Development Kit (page 275)	Hardware
Ethernet Cores		
dwcore_ethernet	10/100 Mbps Operation (page 276)	Synthesizable RTL
dwcore_ethernet_sub	Ethernet MAC Subsystem (page 277)	Synthesizable RTL
dwcore_gig_ethernet	10/100-Mbps and 1-Gbps Operation (page 278)	Synthesizable RTL
dwcore_gig_ethernet_sub	Gigabit Ethernet MAC (GMAC) Subsystem (page 279)	Synthesizable RTL
PCI Cores		
dwcore_pci	32/64 bit 33/66 MHz PCI Core (page 280)	Synthesizable RTL
dwcore_pcix	32/64 bit 133 MHz PCIX Core (page 281)	Synthesizable RTL
dwcore_pci_express	Synthesizable PCI Express Core (page 282)	Synthesizable RTL
USB Cores		
dwcore_usb_otg	USB 2.0 Full Speed On-The-Go Controller Subsystem (page 283)	Synthesizable RTL
dwcore_usb1_device	USB 1.1. Device Controller (page 284)	Synthesizable RTL
dwcore_usb1_host	USB 1.1 OHCI Host Controller (page 285)	Synthesizable RTL
dwcore_usb1_hub	USB 1.1. Hub Controller (page 286)	Synthesizable RTL
dwcore_usb2_host	USB 2.0 Host Controller - UHOST2 (page 287)	Synthesizable RTL
dwcore_usb2_device	USB 2.0 Device Controller (page 288)	Synthesizable RTL
dwcore_usb2_phy	USB 2.0 PHY (page 289)	Hard IP

IEEE 1394 Cores		
dwcore_1394_avlink	IEEE 1394 AVLink (page 290)	Synthesizable RTL
dwcore_1394_device	IEEE 1394 Device Link Controller (page 291)	Synthesizable RTL
dwcore_1394_ohci	IEEE 1394 OHCI Link (page 292)	Synthesizable RTL
dwcore_1394_cphy	IEEE 1394 Cable PHY (page 293)	Synthesizable RTL
Java Acceleration Core		
dwcore_jvxtreme	Java™ Acceleration (page 294)	Synthesizable RTL
JPEG Cores		
dwcore_jpeg_codec	JPEG, CODEC (page 295)	Synthesizable RTL
dwcore_jpeg2_codec	JPEG2000 CODEC (page 296)	Synthesizable RTL
dwcore_jpeg2_encod	JPEG2000 Encoder (page 297)	Synthesizable RTL

Also visit the DesignWare Cores web page at:
http://www.synopsys.com/products/designware/dwcores.html

DesignWare Star IP

Synopsys offers DesignWare Library users the ability to evaluate and design easily at their desktop using the following high-performance, high-value IP cores from leading Star IP providers.

Component Name	Component Description	Component Type
DW_IBM440 (Available Q2'03)	PowerPC 440 Microprocessor Core from IBM (page 300)	Synthesizable RTL[a] Verification Model
DW_V850E-Star	V850E Processor Core from NEC (page 302)	Synthesizable RTL[a] Verification Model
DW_C166S	16-bit Processor from Infineon (page 304)	Synthesizable RTL[a] Verification Model
DW_TriCore1	TriCore1 32-Bit Processor Core from Infineon (page 306)	Synthesizable RTL[a] Verification Model
DW_MIPS4KE	Processor Core Family from MIPS (page 308)	Synthesizable RTL[a] Verification Model
Mezoe Interface Express	Interface Express™ Toolkit including BlueStack® Protocol stack with Profiles and ProtoDeveloper™ Software (page 310)	Software Source Code
SMSC USB 2.0 PHY	Standard Microsystems Corporation GT3100 USB 2.0 PHY Star IP MacroCell (page 311)	Hard IP

a. Verification models of these cores are included in the DesignWare Library. Synthesizable RTL of these cores are available through the Star IP Program.

Also visit the DesignWare Star IP web page at:
http://www.synopsys.com/products/designware/star_ip.html

2 DesignWare Library Synthesizable IP

This chapter briefly describes theDesignWare Library synthesizable IP in the following subsections:

- Building Block IP (Datapath, Data Integrity, Test...)
- AMBA On-Chip Bus Logic and Peripherals IP (page 231)
- Memory IP (Memory BIST, Memory Controller - page 249)
- Bus and I/O Standards (PCI, PCI-X, PCI Express, USB 2.0...)

Building Block IP

The DesignWare Building Block IP (formally called Foundation Library) is a collection of over 140 technology-independent, high-quality, high-performance IP. Most of these IP elements include multiple implementations to provide a variety of performance and area tradeoff options.

DesignWare Building Block IP has superior datapath capabilities based on the Synopsys datapath generator technology. The datapath generator feature of the DesignWare Library automatically produces complex datapath IP elements with better quality of results and significantly shortens synthesis run times.

Since the DesignWare Library is tightly integrated with Synopsys synthesis tools, such as Design Compiler and Physical Compiler, the synthesis tool automatically picks the right architecture with the best speed and area optimizations. In many cases, there is significant improvement on the quality of results automatically because DesignWare infers these arithmetic IP elements without intervention, through Design Compiler.

This library contains high-performance implementations of Basic Library IP plus many IP that implement more advanced arithmetic and sequential logic functions. The DesignWare Building Block IP consists of:

- Basic: A set of IP bundled with HDL Compiler that implements several common arithmetic and logic functions
- Datapath: Arithmetic, trigonometric, and sequential math IP
- Memory: Registers, FIFOs, and FIFO controllers, synchronous and asynchronous RAMs, and stack IP
- Data Integrity: CRC, ECC, 8b10b and other IP
- Test: JTAG IP
- Logic: Combinational, sequential and control IP
- Interface: Debugger IP
- DSP: Digital FIR filter IP
- GTECH: Technology-independent IP library to aid users in developing technology-independent parts

Building Block IP QuickStart

The following topics provide the basic information to get started using the DesignWare Building Block IP.

Updating Building Block IP

To get the latest version and receive the best performance, install the Electronic Software Transfer (EST) release of DesignWare from the following Web site:

http://www.synopsys.com/designware/dwest

If you prefer, you may also send an email to dw_EST@synopsys.com with EST in subject line. In that email, send the following information in the body of the message, in the following format:

<Site Id> <Synopsys Release#>

For example, if your site id is 555 and you want to install the EST for use with the 2000.11 version of the Synopsys Synthesis CD, write the following two fields in the body of the message separated by a few blank spaces:

```
555   2000.11
```

Setting Up DesignWare Building Block IP

Include the following lines in your .synopsys_dc.setup file and ensure that you have a valid DesignWare license:

```
synthetic_library = {dw_foundation.sldb}
link_library = target_library + synthetic_library
search_path = search_path + {synopsys_root + "/dw/sim_ver"}
synlib_wait_for_design_license = {"DesignWare"}
```

Accessing DesignWare Building Block IP

You can access DesignWare Building Block IP either by operator or functional inference, or by instantiating the component directly. The example below shows how to access these IP:

```
Verilog
assign PROD = IN1 * IN2; // Operator Inference
assign PROD = mult_tc(IN1, IN2); // Function Inference
DW01_mult #(8, 8) U1 (A, B, TC, PRODUCT); // Instantiation
```

Details about inference and instantiation in VHDL and Verilog are in the following directory: $SYNOPSYS/dw/examples.

Synthesizing DesignWare Building Block IP

Design Compiler automatically selects the best implementation for combinational DesignWare Building Block IP. You can also force Design Compiler to select the implementation of your choice either by adding Synopsys Compiler directives or by using the following commands:

```
dc_shell> set_dont_use standard.sldb/DW01_add/rpl
dc_shell> set_implementation clf {add_68}
```

Simulating DesignWare Building Block IP

Synopsys VCS simulator uses the default setup file while simulating DesignWare Building Block IP. Use the following options to simulate DesignWare Building Block IP with a Verilog simulator:

```
-y $SYNOPSYS/dw/sim_ver +libext+.v+
```

Technical Support or Further Information

For information on how to contact us, refer to “Getting Help” on page 13.

Building Block IP in FPGA Compiler II QuickStart

The following topics provide the basic information to get started using the DesignWare Building Block IP with FPGA Compiler II.

Updating FPGA Compiler II

FPGA Compiler II versions 3.2 and later support instantiated DesignWare Building Block IP. Install the latest release of FPGA Compiler II to get the best performance as well as access to the latest FPGA technologies. FPGA Compiler II customers who are on active maintenance will automatically receive CDs for the latest major release, or from any Synopsys sales office or at the following location:
http://www.synopsys.com/products/fpga

Setting Up DesignWare Building Block IP in FPGA Compiler II

DesignWare Building Block IP are automatically installed by default during the installation of FPGA Compiler II versions 3.2 and later. You can choose to not install DesignWare Building Block IP by unchecking DesignWare in the FPGA Vendors dialog box during installation. Otherwise, there is nothing equivalent to the.synopsys_dc.setup file for Design Compiler to modify.

License Requirement

FPGA Compiler II versions 3.2 to 3.3 require a valid DesignWare license in order to implement all DesignWare Building Block IP. Beginning in FPGA Compiler II version 3.5, DesignWare Building Block basic IP can be implemented without the requirement of a DesignWare license. The basic IP include the following:

DW01_cmp2	DW01_cmp6	DW01_absval	DW01_add	DW01_sub
DW01_addsub	DW01_inc	DW01_dec	DW01_incdec	DW02_mult

Accessing DesignWare Building Block IP in FPGA Compiler II

You can access DesignWare Building Block IP in FPGA Compiler II versions 3.2 and later by direct instantiation. For example:

In Verilog:

```
DW02_mult #(inst_A_width, inst_B_width)
  U1 (.A(inst_A), .B(inst_B), .TC(inst_TC), .PRODUCT(inst_PRODUCT));
```

In VHDL:

```
U1: DW02_mult
  generic map ( A_width => inst_A_width, B_width => inst_B_width )
  port map (A => inst_A, B => inst_B, TC => inst_TC,
            PRODUCT => inst_ PRODUCT);
```

Currently FPGA Compiler II does not support inference of DesignWare Building Block IP.

Synthesizing DesignWare Building Block IP in FPGA Compiler II

FPGA Compiler II versions 3.2 and later automatically select the implementation for the chosen FPGA technology. It understands and takes advantage of vendor-specific architectures to provide the best quality of results (QoR) for most DesignWare Building Block IP. Note that some DesignWare Building Block IP are implemented using generic gates but improvement in QoR can be expected in future releases.

Simulating DesignWare Building Block IP

FPGA Compiler II has the ability to generate synthesized netlists in Verilog and VHDL. Just right-click on the optimized chip, select Export Netlist, and then select Verilog or VHDL as the desired output format. The netlists generated are structural netlists which can be simulated with VCS or VSS.

Further Information

For further information on using DesignWare in FPGA Compiler II:

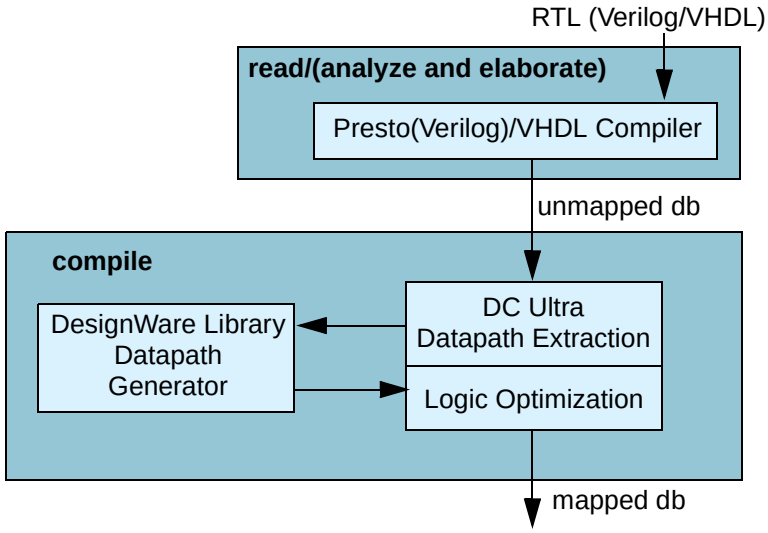
- Visit our Web site at http://www.synopsys.com/products/fpga/fpga_solution.html
- e-mail the Synopsys Support Center at support_center@synopsys.com
- Call (800) 245-8005 (toll free in the United States)

Datapath Generator Overview

The new datapath generators improve the quality of synthesized datapaths in two steps:

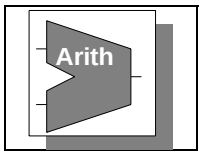
1. By using more sophisticated extraction and partitioning of datapaths from RTL code,
2. By improved synthesis of the extracted datapaths.

The following figure shows the flow for datapath synthesis. After the RTL code is analyzed and elaborated by the Presto(Verilog)/VHDL Compiler, the datapath portions of the RTL are extracted by DC Ultra and then synthesized by the datapath generators in the DesignWare Library.



DC Ultra partitions datapaths that are extracted from RTL into large sum-of-product and product-of-sum blocks. This reduces the number of expensive carry propagations to a minimum, resulting in faster and smaller circuits. In sum-of-products a multiplication can be followed by an addition without a carry propagation before the addition. Similarly, in product-of-sums an addition can be followed by a multiplication without a carry propagation before the multiplication. The same techniques are also applied to reduce the number of carry propagations in magnitude comparisons of complete sum-of-products. Resource and common subexpression sharing allow for further area savings.

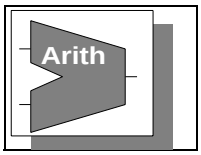
The datapath generators then perform a constraint- and technology-driven synthesis of the extracted sum-of-product and product-of-sum blocks. Enhanced algorithms are used to construct optimized adder reduction trees and carry-propagate adders to meet the given timing constraints with minimal area requirements for the specified technology and conditions. A smart generation feature selects the best among alternative implementation variants. Special datapath library cells are automatically used where available and beneficial. Optimized structures are generated for special arithmetic operations, like constant multiplication or squaring.



Datapath – Arithmetic Overview

The datapath arithmetic DesignWare Building Block IP, many of which are inferred, are applicable to ASIC or FPGA designs. These IP are high-performance arithmetic implementations (based on a fast carry look-ahead architecture) to augment those in the Basic IP Library. The Basic IP Library is included in your (V)HDL Compiler product.

Most IP in this category have multiple architectures for each function (architecturally optimized for either performance or area). This provides you with the best architecture for your design goals. All IP have a parameterized word length.



DW01_absval

Absolute Value

- Parameterized word length

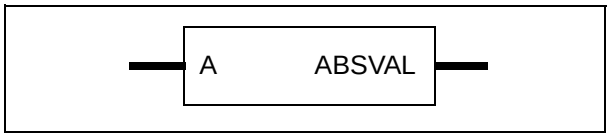


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
ABSVAL	*width* bit(s)	Output	Absolute value of A

Table 2: Parameter Description

Parameter	Values	Function
width	≥ 1	Word length of A and ABSVAL

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW01_add

Adder

- Parameterized word length
- Carry-in and carry-out signals
- Module Compiler Architectures

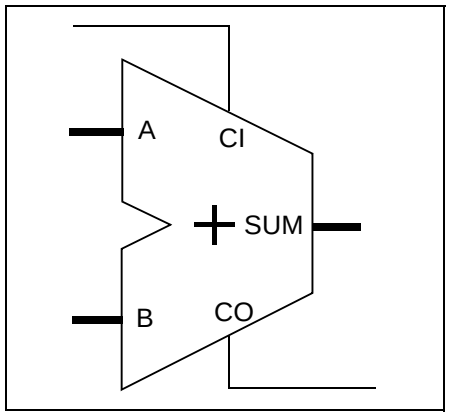


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
CI	1 bit	Input	Carry-in
SUM	*width* bit(s)	Output	Sum of (A + B + CI)
CO	1 bit	Output	Carry-out

Table 2: Parameter Description

Parameter	Values	Description
width	≥1	Word length of A, B, and SUM

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation
bk	Brent-Kung architecture synthesis model	DesignWare-Foundation
csm[b]	Conditional-sum synthesis model	DesignWare-Foundation
rpcs	Ripple-carry-select architecture	DesignWare-Foundation
clsa[c]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation
csa[c]	MC-inside-DW carry-select	DesignWare-Foundation
fastcla[c]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation

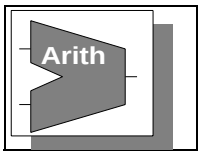


Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
pprefix[c]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The performance of the csm implementation is heavily dependent on the use of a high-performance inverting 2-to-1 multiplexer in the technology library. In such libraries, the csm implementation exhibits a superior area-delay product. Although the csm implementation does not always surpass the delay performance of the clf implementation, it is much lower in area.

c. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable ‘dw_prefer_mc_inside’ must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW01_addsub

Adder-Subtractor

- Parameterized word length
- Carry-in and carry-out signals

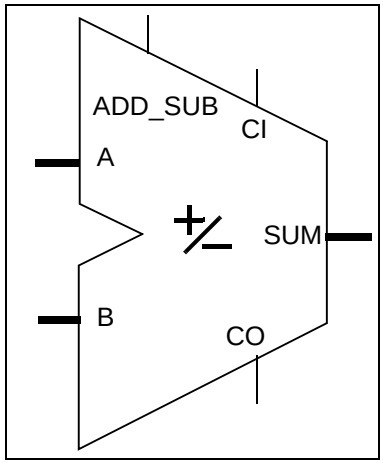


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
CI	1 bit	Input	Carry/borrow-in
ADD_SUB	1 bit	Input	Addition/subtraction control
SUM	*width* bit(s)	Output	Sum (A +B +CI) or difference (A –B –CI)
CO	1 bit	Output	Carry/borrow-out

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A, B, and SUM

Table 3: Synthesis Implementations[a]

Implementation Name	Implementation	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
bk	Brent-Kung architecture synthesis model	DesignWare-Foundation
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation
csm[b]	Conditional-sum synthesis model	DesignWare-Foundation
rpcs	Ripple-carry-select architecture	DesignWare-Foundation
clsa[c]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation

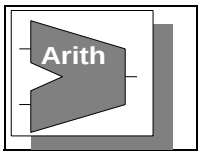


Table 3: Synthesis Implementations[a] (Continued)

Implementation Name	Implementation	License Required
csa[c]	MC-inside-DW carry-select	DesignWare-Foundation
fastcla[c]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation
pprefix[c]	MC-inside-DW flexible parallel prefix	DesignWare-Foundation

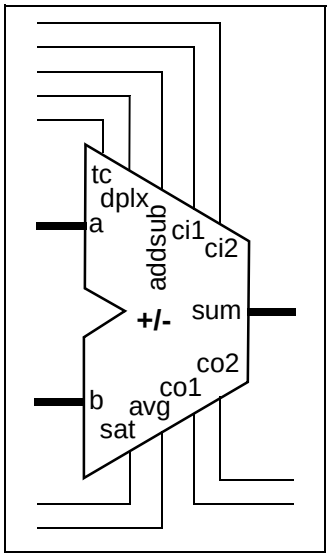
a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The performance of the csm implementation is heavily dependent on the use of a high-performance inverting 2-to-1 MUX in the technology library. In such libraries, the csm implementation exhibits a superior area-delay product. Although the csm implementation does not always surpass the delay performance of the clf implementation, it is much lower in area.

c. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable ‘dw_prefer_mc_inside’ must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW_addsub_dx

Duplex Adder/Subtractor with Saturation and Rounding



- Selectable single full-width Add/Sub (simplex) or two smaller width Add/Sub operations (duplex)
- Selectable saturation mode
- Selectable average mode
- Selectable number system (unsigned or twos complement)
- Parameterized full word width
- Parameterized partial word width (allowing for asymmetric partial width operations)
- Carry-out signals (one for lower half and one for full and upper half) that numerically extend the calculated sum (maintaining full precision)
- Carry-in signals (one for full and lower half and one for upper half)

Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Input data
b	*width* bit(s)	Input	Input data
ci1	1 bit	Input	Full or part1 carry input
ci2	1 bit	Input	Part2 carry input
addsub	1 bit	Input	Add/subtract select input 0 = performs add 1 = performs subtract
tc	1 bit	Input	Two’s complement select (active high)
sat	1 bit	Input	Saturation mode select (active high)
avg	1 bit	Input	Average mode select (active high)
dplx	1 bit	Input	Duplex mode select (active high)
sum	*width* bit(s)	Output	Output data

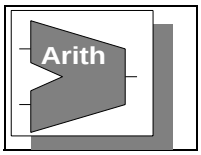


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
co1	1 bit	Output	Part1 carry output
co2	1 bit	Output	Full width or part2 carry output

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 4	Word width of a, b, and sum
p1_width	2 to *width*–2	Word width of part1 of duplex Add/Sub

Table 3: Synthesis Implementations [a]

Implementation Name	Function	License Required
rpl	Ripple Carry Synthesis Model	DesignWare-Foundation
rpcs	Ripple Carry Select Synthesis Model	DesignWare-Foundation
csm	Conditional Sum Synthesis Model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW01_ash
Arithmetic Shifter

- Parameterized word length
- Parameterized shift coefficient width
- Inferable using a function call

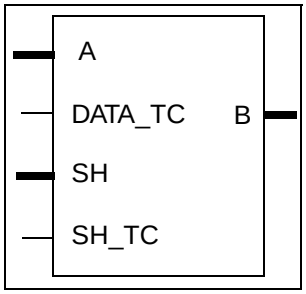


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Input data
DATA_TC	1 bit	Input	Data two's complement control 0 = unsigned 1 = signed
SH	*SH_width* bit(s)	Input	Shift control
SH_TC	1 bit	Input	Shift two's complement control 0 = unsigned 1 = signed
B	*A_width* bit(s)	Output	Output data

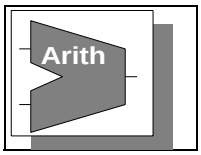
Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 2	Word length of A and B
SH_width	≥ 1	Word length of SH

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
mx2	Implement using 2:1 multiplexers only	DesignWare-Foundation
mx2i	Implement using 2:1 inverting multiplexers and 2:1 multiplexers	DesignWare-Foundation
mx4	Implement using 4:1 and 2:1 multiplexers	DesignWare-Foundation
mx8	Implement using 8:1, 4:1, and 2:1 multiplexers	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_bin2gray

Binary to Gray Converter

- Parameterized word length
- Inferable using a function call

Table 1: Pin Description

Pin Name	Width	Direction	Function
b	*width* bit(s)	Input	Binary coded input data
g	*width* bit(s)	Output	Gray coded output data

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Input word length

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW01_bsh

Barrel Shifter

- Parameterized data and shift coefficient word lengths
- Inferable using a function call

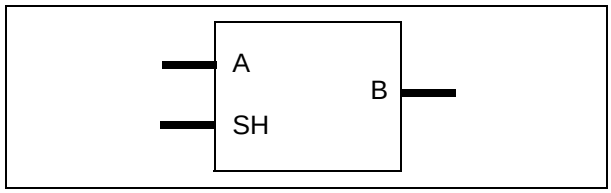


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width*	Input	Input data
SH	*SH_width*	Input	Shift control
B	*A_width*	Output	Shifted data out

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A and B
SH_width	≤ceil($\log_2$[*A_width*]) for mx2, mx2i ≥ 1 for mx4, mx8	Word length of SH

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
mx2	Implement using 2:1 multiplexers only	DesignWare-Foundation
mx2i	Implement using 2:1 inverting multiplexers and 2:1 multiplexers	DesignWare-Foundation
mx4	Implement using 4:1 and 2:1 multiplexers	DesignWare-Foundation
mx8	Implement using 8:1, 4:1, and 2:1 multiplexers	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

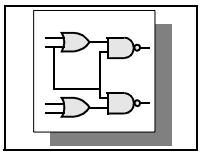


Table 4: Sample Parameter Values

A_width	SH_width
1 - 2	1
3 - 4	2
5 - 8	3
9 - 16	4
17 - 32	5
33 - 64	6

DW01_cmp2

2-Function Comparator

- Parameterized word length
- Unsigned and signed (two's-complement) data operation

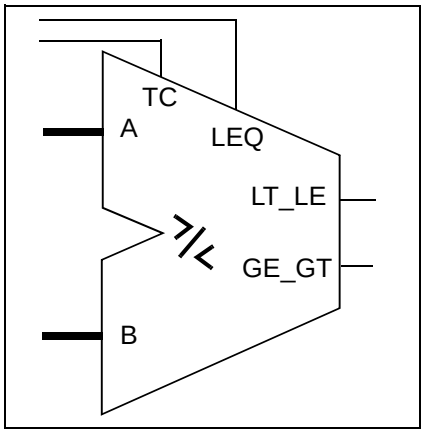


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
LEQ	1 bit	Input	Output condition control
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
LT_LE	1 bit	Output	Less-than/less-than-or-equal output condition
GE_GT	1 bit	Output	Greater-than-or-equal/greater-than output condition

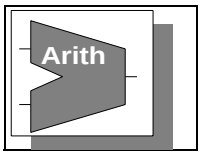
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A and B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
bk	Brent-Kung synthesis model	DesignWare-Foundation
cla	Carry-look-ahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW01_cmp6

6-Function Comparator

- Parameterized word length
- Unsigned and signed (two's-complement) data comparison

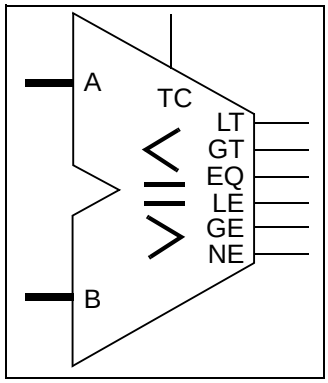


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
LT	1 bit	Output	Less-than output condition
GT	1 bit	Output	Greater-than output condition
EQ	1 bit	Output	Equal output condition
LE	1 bit	Output	Less-than-or-equal output condition
GE	1 bit	Output	Greater-than-or-equal output condition
NE	1 bit	Output	Not equal output condition

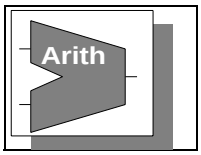
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A and B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
bk	Brent-Kung synthesis model	DesignWare-Foundation
cla	Carry-look-ahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_cmp_dx

Duplex Comparator

- Selectable single full width Compare, or two smaller width Compare operations (duplex)
- Selectable number system (unsigned or two's complement)
- Parameterized full word width
- Parameterized partial word width (allowing for asymmetric partial width operations)
- Separate flags for Less Than, Equal To, and Greater Than
- Two sets of flags for duplex operation

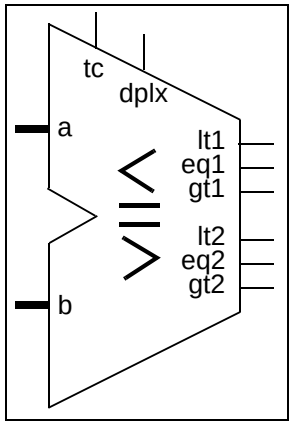


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Input data
b	*width* bit(s)	Input	Input data
tc	1 bit	Input	Two's complement control
dplx	1 bit	Input	Duplex mode select (active high)
lt1	1 bit	Output	Part1 : less-than output condition
eq1	1 bit	Output	Part1 : equal output condition
gt1	1 bit	Output	Part1 : greater-than output condition
lt2	1 bit	Output	Full width or part2 : less-than output condition
eq2	1 bit	Output	Full width or part2 : equal output condition
gt2	1 bit	Output	Full width or part2 : greater-than output condition

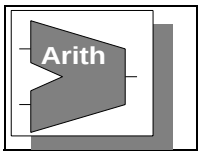
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 4	Word width of a and b
p1_width	2 to *width*–2	Word width of part1 of duplex compare

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
bk	Brent-Kung synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_cntr_gray

Gray Code Counter

- Gray encoded output
- Asynchronous and synchronous reset
- Count enable

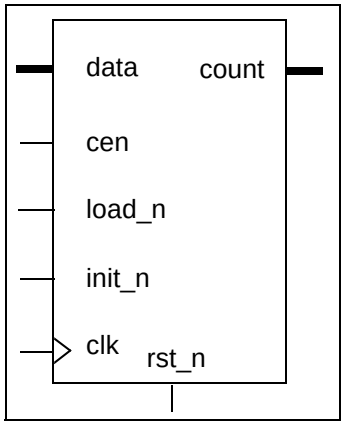


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, asynchronous, active low
init_n	1 bit	Input	Reset, synchronous, active low
load_n	1 bit	Input	Enable data load to counter, active low
data	*width* bit(s)	Input	Counter load input
cen	1 bit	Input	Count enable, active high
count	*width* bit(s)	Output	Gray coded counter output

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of counter

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	DesignWare-Foundation
cla	Carry-lookahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW01_csa

Carry Save Adder

- Parameterized word length
- Carry-in and carry-out signals

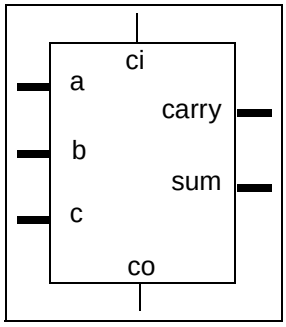


Table 1: Pin Description

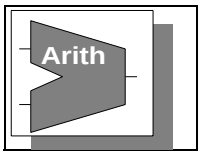
Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Input data
b	*width* bit(s)	Input	Input data
c	*width* bit(s)	Input	Input data
ci	1 bit	Input	Carry-in
carry	*width* bit(s)	Output	Carry output data
sum	*width* bit(s)	Output	Sum output data
co	1 bit	Output	Carry-out

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of a, b, c, sum, and carry

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW01_dec
Decrementer

- Parameterized word length

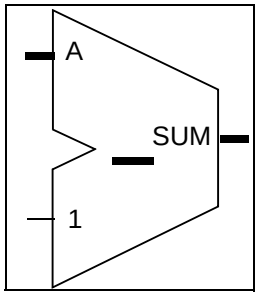


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
SUM	*width* bit(s)	Output	Decremented (A –1)

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A and SUM

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation
clsa[b]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation
csa[b]	MC-inside-DW carry-select	DesignWare-Foundation
fastcla[b]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation
pprefix[b]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW_div
Combinational Divider

- Parameterized word lengths
- Unsigned and signed (two's complement) data operation
- Remainder or modulus as second output

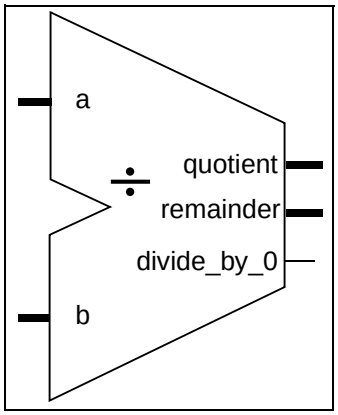


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*a_width* bit(s)	Input	Dividend
b	*b_width* bit(s)	Input	Divisor
quotient	*a_width* bit(s)	Output	Quotient
remainder	*b_width* bit(s)	Output	Remainder / modulus
divide_by_0	1 bit	Output	Indicates if b equals 0

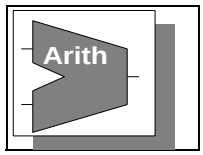
Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 2 Default: None	Word length of a
b_width	≥ 2, ≤*a_width* Default: None	Word length of b
tc_mode	0 or 1 Default: 0	Two'- complement control
rem_mode	0 or 1 Default: 1	Remainder output control

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Restoring ripple-carry synthesis model	DesignWare-Foundation
cla	Restoring carry-look-ahead synthesis model	DesignWare-Foundation
cla2	Restoring carry-look-ahead, 2-way overlapped synthesis model	DesignWare-Foundation

DW_div
Combinational Divider



a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_div_pipe

Stallable Pipelined Divider

- Parameterized word length
- Parameterized unsigned and signed data operation
- Parameterized number of pipeline stages
- Parameterized stall mode (stallable or non-stallable)
- Parameterized reset mode (no reset, asynchronous or synchronous reset)
- Automatic pipeline retiming

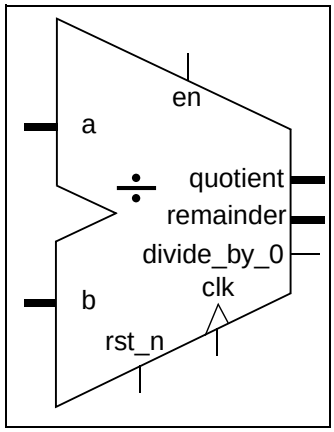


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset, active-low (not used if parameter rst_mode=0)
en	1 bit	Input	Load enable (used only if parameter stall_mode=1) 0 = stall 1 = load
a	*a_width* bit(s)	Input	Dividend
b	*a_width* bit(s)	Input	Divisor
quotient	*a_width* bit(s)	Output	Quotient a / b
remainder	*b_width* bit(s)	Output	Remainder

Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 2 Default: None	Word length of a
b_width	≥ 2 ≤*a_width* Default: None	Word length of b
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = signed

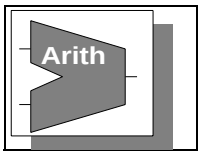


Table 2: Parameter Description (Continued)

Parameter	Values	Description
rem_mode	0 or 1 Default: 1	Remainder output control 0 = modulus 1 = remainder
num_stages	≥ 2 Default: 2	Number of pipeline stages
stall_mode	0 or 1 Default: 1	Stall mode 0= non-stallable 1 = stallable
rst_mode	0 to 2 Default: 1	Reset mode 0 = no reset 1 = asynchronous reset 2 = synchronous reset

Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str[a]	Pipelined str synthesis model	DesignWare-Foundation

a. One of rpl, cla or cl2 implementation is selected based the constraints of the design.

DW_gray2bin

Gray-to-Binary Converter

- Parameterized word length
- Inferable using a function call

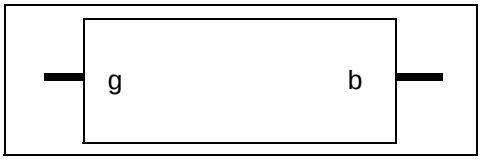


Table 1: Pin Description

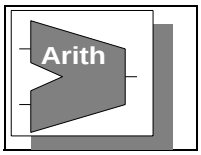
Pin Name	Width	Direction	Function
g	*width* bit(s)	Input	Gray coded input data
b	*width* bit(s)	Output	Binary coded output data

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Input word length

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	DesignWare-Foundation
cla	Carry-lookahead synthesis model	DesignWare-Foundation



DW01_inc
Incrementer

- Parameterized word length

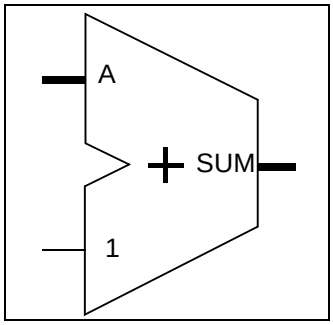


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
SUM	*width* bit(s)	Output	Increment (A +1)

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A and SUM

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation
clsa[b]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation
csa[b]	MC-inside-DW carry-select	DesignWare-Foundation
fastcla[b]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation
pprefix[b]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW01_incdec

Incrementer-Decrementer

- Parameterized word length

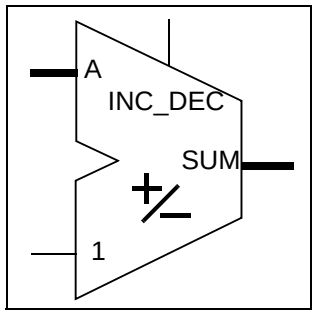


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
INC_DEC	1 bit	Input	Increment control 0 = increment (A + 1) 1 = decrement (A –1)
SUM	*width* bit(s)	Output	Increment (A +1) or decrement (A –1)

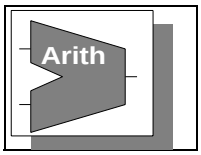
Table 2: Parameter Description

Parameter	Values	Function
width	≥ 1	Word length of A and SUM

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	none
cla	Carry look-ahead synthesis model	none
clf	Fast carry look-ahead synthesis model	DesignWare-Foundation
clsa[b]	MC inside DW carry-look-ahead-select	DesignWare-Foundation
csa[b]	MC inside DW carry-select	DesignWare-Foundation
fastcla[b]	MC inside DW fast carry-look-ahead	DesignWare-Foundation
pprefix[b]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



b. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW_inc_gray

Gray Incrementer

- Parameterized word length
- Inferable using a function call

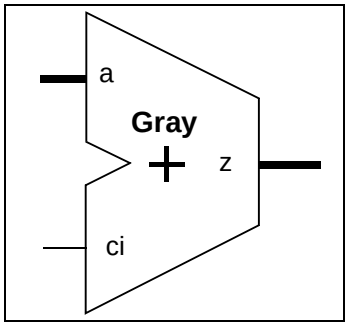


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Gray coded input data
ci	1 bit	Input	Carry-in
z	*width* bit(s)	Output	Gray coded output data

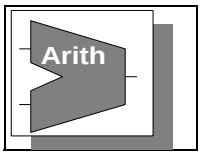
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Input word length

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	DesignWare-Foundation
cla	Carry-lookahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW02_mac

Multiplier-Accumulator

- Parameterized word length
- Unsigned and signed (two's-complement) data operation

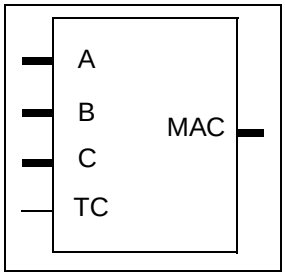


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
C	*A_width* + *B_width* bit(s)	Input	Addend
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
MAC	*A_width* + *B_width* bit(s)	Output	MAC result (A × B + C)

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1	Word length of B

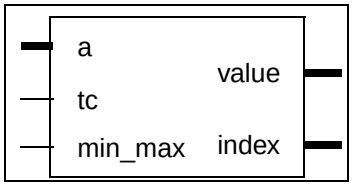
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa	Carry-save array synthesis model	DesignWare-Foundation
wall	Booth-recoded Wallace tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_minmax

Minimum/Maximum Value



- Parameterized number of inputs
- Parameterized word length
- Unsigned and signed (two's complement) data operation
- Dynamically selectable mode (minimum or maximum)
- Additional output gives an index of the minimum or maximum input
- Inferable using a function call

Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*num_inputs* × *width* bit(s)	Input	Concatenated input data
tc	1 bit	Input	Two's complement control
min_max	1 bit	Input	Minimum/maximum control 0 = minimum (a) 1 = maximum (a)
value	*width* bit(s)	Output	Minimum/maximum value
index	ceil($\log_2$[*num_inputs*]) bit(s)	Output	Index of minimum/maximum input

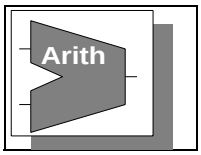
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Input word length
num_inputs	≥ 2 Default: 2	Number of inputs

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
cla	Carry-lookahead tree synthesis model	DesignWare-Foundation
clas	Carry-lookahead/select tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW02_mult

Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation

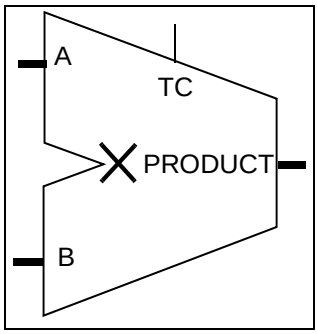


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product A × B

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa	Carry-save array synthesis model	none
nbw	Non-Booth-recoded Wallace-tree synthesis model[b]	DesignWare-Foundation
wall	Booth-recoded Wallace-tree synthesis model[c]	DesignWare-Foundation
mcarch[de]	MC-inside-DW Wallace-tree	DesignWare-Foundation
csmult[de]	MC-inside-DW flexible Booth Wallace	DesignWare-Foundation

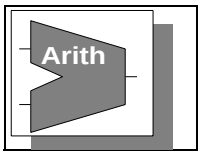
a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. In most cases, the nbw implementation generates the fastest and smallest circuits for small- to medium-sized multipliers. For multipliers having products larger than 41 bits (such as, *A_width+B_width* > 41) the nbw implementation would prove of no benefit. So, for products larger than 41 bits, the nbw implementation will produce a Booth recoded multiplier identical to the wall implementation.

c. In most cases, the wall implementation generates faster and smaller circuits for medium- to large-sized multipliers.

d. Automatically selects Booth-recoding or non-Booth-recoding, depending on constraints.

e. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.



DW02_multp

Partial Product Multiplier

- Parameterized word lengths
- Parameterized sign extension of partial product outputs for use in summing products
- Unsigned and signed (two's-complement) data operation

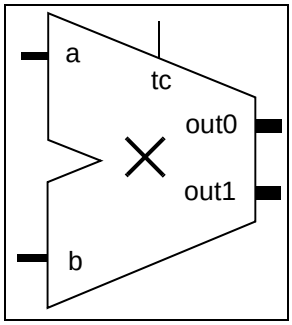


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*a_width* bit(s)	Input	Multiplier
b	*b_width* bit(s)	Input	Multiplicand
tc	1 bit	Input	Two's complement 0 = unsigned 1 = signed
out0	*out_width* bit(s)	Output	Partial product of (a × b)
out1	*out_width* bit(s)	Output	Partial product of (a × b)

Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 1	Word length of a
b_width	≥ 1	Word length of b
out_width	≥ *a_width* + *b_width* + 2	Word length of out0 and out1

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
wall	Booth-recoded Wallace tree synthesis model [b]	DesignWare-Foundation
nbw	Non-Booth-recoded Wallace tree synthesis mdl[c]	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. In most cases, the wall implementation generates both faster and smaller circuits for medium- to large-sized multipliers.

c. In most cases, the nbw implementation generates both faster and smaller circuits for small- to medium-sized multipliers.

DW02_mult_2_stage

Two-Stage Pipelined Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation
- Two-stage pipelined architecture
- Automatic pipeline retiming
- Inferable from Behavioral Compiler

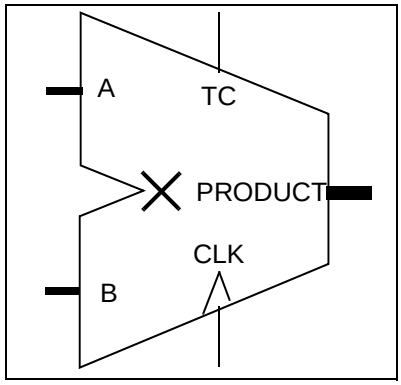


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
CLK	1 bit	Input	Clock
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product (A × B)

Table 2: Parameter Description

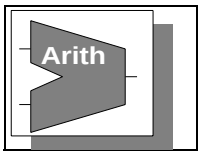
Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1 (For csa architecture: *A_width* +*B_width* ≤48)	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa[b]	Carry-save array synthesis model	DesignWare-Foundation
str	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The csa implementation is only valid when the sum of *A_width* and *B_width* ≤48 bits, as it has no area benefit beyond 48 bits.



DW02_mult_3_stage

Three-Stage Pipelined Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation
- Three-stage pipelined architecture
- Automatic pipeline retiming
- Inferable from Behavioral Compiler

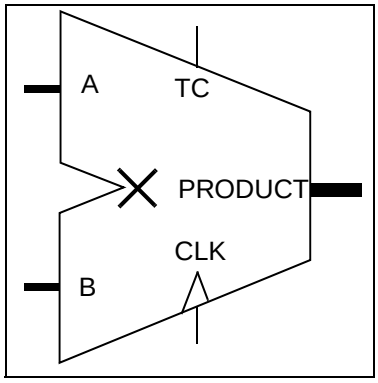


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
CLK	1 bit	Input	Clock
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product (A × B)

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1 (For csa architecture: *A_width* +*B_width* ≤48)	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa[b]	Carry-save array synthesis model	DesignWare-Foundation
str	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The csa implementation is only valid when the sum of *A_width* and *B_width* ≤48 bits, as it has no area benefit beyond 48 bits.

DW02_mult_4_stage

Four-Stage Pipelined Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation
- Four-stage pipelined architecture
- Automatic pipeline retiming
- Inferable from Behavioral Compiler

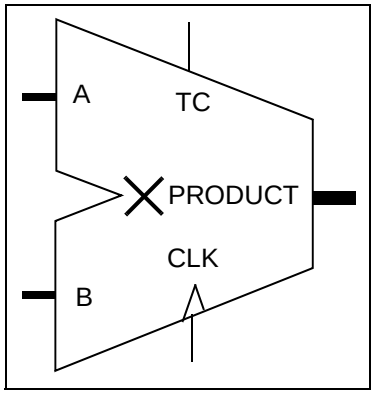


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
CLK	1 bit	Input	Clock
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product (A × B)

Table 2: Parameter Description

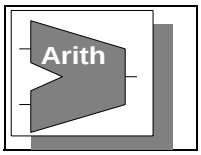
Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1 (For csa architecture: *A_width* +*B_width* ≤48)	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa[b]	Carry-save array synthesis model	DesignWare-Foundation
str	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The csa implementation is only valid when the sum of *A_width* and *B_width* ≤48 bits, as it has no area benefit beyond 48 bits.



DW02_mult_5_stage

Five-Stage Pipelined Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation
- Five-stage pipelined architecture
- Automatic pipeline retiming
- Inferable from Behavioral Compiler

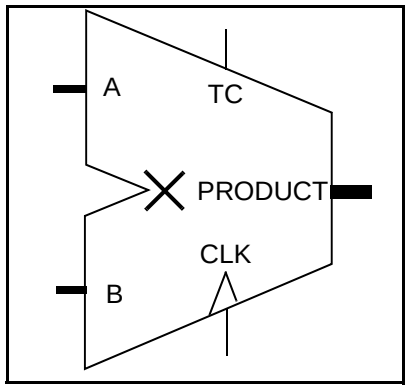


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement 0 = unsigned 1 = signed
CLK	1 bit	Input	Clock
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product (A × B)

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1 (For csa architecture: *A_width* +*B_width* ≤48)	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa[b]	Carry-save array synthesis model	DesignWare-Foundation
str	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The csa implementation is only valid when the sum of *A_width* and *B_width* ≤48 bits, as it has no area benefit beyond 48 bits.

DW02_mult_6_stage

Six-Stage Pipelined Multiplier

- Parameterized word length
- Unsigned and signed (two's-complement) data operation
- Six-stage pipelined architecture
- Automatic pipeline retiming
- Inferable from Behavioral Compiler

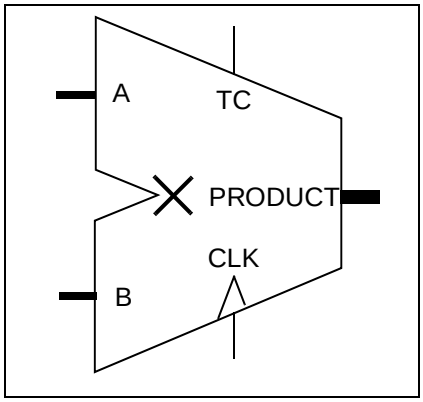


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement 0 = unsigned 1 = signed
CLK	1 bit	Input	Clock
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product (A × B)

Table 2: Parameter Description

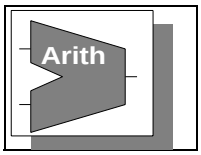
Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1 (For csa architecture: *A_width* +*B_width* ≤48)	Word length of B

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa[b]	Carry-save array synthesis model	DesignWare-Foundation
str	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. The csa implementation is only valid when the sum of *A_width* and *B_width* ≤48 bits, as it has no area benefit beyond 48 bits.



DW_mult_dx

Duplex Multiplier

- Selectable single full-width multiplier (simplex) or two parallel smaller-width multiplier (duplex) operations
- Area and delay are similar to those of the DW02_mult wallace architecture
- Selectable number system (unsigned or two's complement)
- Parameterized full word width
- Parameterized partial word width (allowing for asymmetric partial width operations)

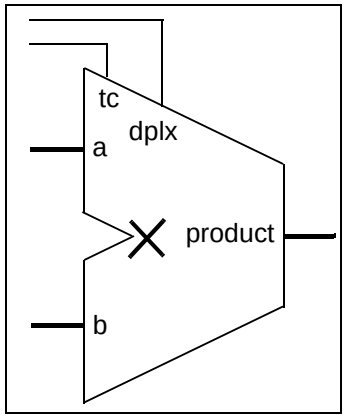


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Input data
b	*width* bit(s)	Input	Input data
tc	1 bit	Input	Two's complement control
dplx	1 bit	Input	Duplex mode select, active high
product	*width* × 2 bit(s)	Output	Product(s)

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 4[a]	Word width of a and b
p1_width	2 to *width*–2[b]	Word width of Part1 of duplex multiplier

a. Due to the limitation of memory addressing ranges of the computer operating system, there is an upper limit for parameter *width*.

b. For the best performance of DW_mult_dx, *p1_width* should be set in the range [*width*/2, *width*–2].

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
wall	Booth-recoded Wallace-tree synthesis model	DesignWare-Foundation

DW_mult_pipe

Stallable Pipelined multiplier

- Parameterized word length
- Unsigned and signed (two's complement) pipelined multiplication
- Parameterized number of pipeline stages
- Parameterized stall mode (stallable or non-stallable)
- Parameterized reset mode (no reset, asynchronous or synchronous reset)
- Automatic pipeline retiming

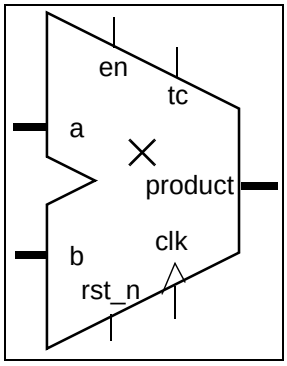


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset, active-low (not used if parameter rst_mode=0)
en	1 bit	Input	Load enable (used only if parameter stall_mode=1) 0 = stall 1 = load
tc	1 bit	Input	Two's complement control: 0 = unsigned 1 = signed
a	*a_width* bit(s)	Input	Multiplier
b	*b_width* bit(s)	Input	Multiplicand
product	*a_width*+*b_width* bit(s)	Output	Product a × b

DW_mult_pipe
Stallable Pipelined multiplier

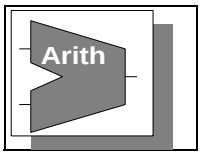


Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 1 Default: None	Word length of a
b_width	≥ 1 Default: None	Word length of b
num_stages	≥ 2 Default: 2	Number of pipeline stages
stall_mode	0 or 1 Default: 1	Stall mode 0 = non-stallable 1 = stallable
rst_mode	0 to 2 Default: 1	Reset mode 0 = no reset 1 = asynchronous reset 2 = synchronous reset

Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str[a]	Pipelined str synthesis model	DesignWare-Foundation

a. One of csa, wall or nbw implementation is selected based on the constraints of the design.

DW02_prod_sum

Generalized Sum of Products

- Parameterized number of inputs
- Parameterized word length

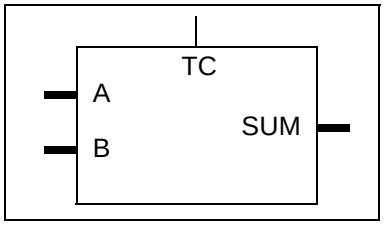


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* × *num_inputs* bit(s)	Input	Concatenated input data
B	*B_width* × *num_inputs* bit(s)	Input	Concatenated input data
TC	1 bit	Input	Two's complement 0 = unsigned 1 = signed
SUM	*SUM_width* bit(s)	Output	Sum of products

Table 2: Parameter Description

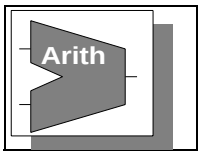
Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1[a]	Word length of B
num_inputs	≥ 1	Number of inputs
SUM_width	≥ 1	Word length of SUM

a. For nbw implementation, *A_width*+*B_width* ≤36. Due to concern of implementation selection run time, a limitation is set for *A_width* and *B_width*.

Table 3: Synthesis Implementations[a]

Implementation	Function	License Required
csa	Carry-save array synthesis model	DesignWare-Foundation
wall	Booth-recoded Wallace-tree synthesis model[b]	DesignWare-Foundation
nbw	Non-Booth-recoded Wallace-tree synthesis model[c]	DesignWare-Foundation
mcarch[de]	MC-inside-DW Wallace-tree	DesignWare-Foundation
csmult[de]	MC-inside-DW flexible Booth Wallace	DesignWare-Foundation

DW02_prod_sum
Generalized Sum of Products



a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. In most cases the wall implementation generates both faster and smaller circuits for medium- to large-sized multipliers.

c. In most cases the nbw implementation generates both faster and smaller circuits for small- to medium-sized multipliers.

d. Automatically chooses Booth-recoded or non-Booth-recoded architectures, depending on constraints.

e. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW02_prod_sum1

Multiplier-Adder

- Parameterized number of inputs
- Parameterized word length

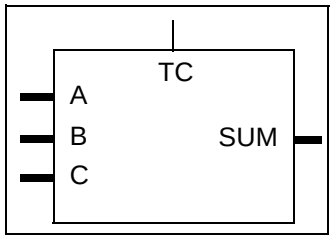


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Input data
B	*B_width* bit(s)	Input	Input data
C	*SUM_width* bit(s)	Input	Input data
TC	1 bit	Input	Two's complement 0 = unsigned 1 = signed
SUM	*SUM_width* bit(s)	Output	Sum of products

Table 2: Parameter Description

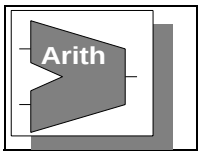
Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1[a]	Word length of B
SUM_width	≥ 1	Word length of C and output SUM

a. For nbw implementation, *A_width*+*B_width* ≤36. Due to concern of implementation selection run time, a limitation is set for *A_width* and *B_width*.

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa	Carry-save array synthesis model	DesignWare-Foundation
wall	Booth-recoded Wallace-tree synthesis model[b]	DesignWare-Foundation
nbw	Non-Booth-recoded Wallace-tree synthesis model[c]	DesignWare-Foundation

DW02_prod_sum1
Multiplier-Adder



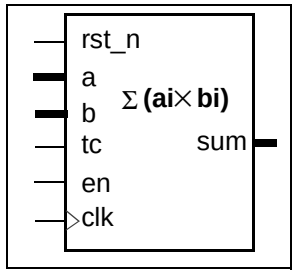
a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. In most cases, the wall implementation generates both faster and smaller circuits for medium- to large-sized multipliers.

c. In most cases the nbw implementation generates both faster and smaller circuits for small- to medium-sized multipliers.

DW_prod_sum_pipe

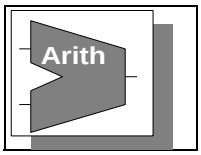
Stallable Pipelined Generalized Sum of Products



- Parameterized word length
- Unsigned and signed (two's complement) data operation
- Parameterized number of pipeline stages
- Parameterized stall mode (stallable or non-stallable)
- Parameterized reset mode (no reset, asynchronous or synchronous reset)
- Automatic pipeline retiming

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset, active-low (not used if parameter rst_mode=0)
en	1 bit	Input	Load enable (used only if parameter stall_mode=1) 0 = stall 1 = load
tc	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
a	*a_width* × *num_inputs* bit(s)	Input	Concatenated input data vector
b	*b_width* × *num_inputs* bit(s)	Input	Concatenated input data vector
sum	*sum_width* bit(s)	Output	Pipelined data summation



DW_prod_sum_pipe
Stallable Pipelined Generalized Sum of Products

Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 1 Default: None	Word length of a
b_width	≥ 1 Default: None	Word length of b
num_inputs	>1 Default: 2	Number of inputs
num_stages	≥ 2 Default: 2	Number of pipeline stages
stall_mode	0 or 1 Default: 1	Stall mode 0 = non-stallable 1 = stallable
rst_mode	0 to 2 Default: 1	Reset mode 0 = no reset 1 = asynchronous reset 2 = synchronous reset
sum_width	≥ 1 Default: None	Word length of sum

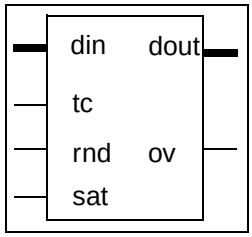
Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str[a]	Pipelined str synthesis model	DesignWare-Foundation

a. One of csa, wall or nbw implementation is selected based on the constraints of the design.

DW01_satrnd

Arithmetic Saturation and Rounding Logic



- Parameterized word length
- Dynamically or statically configurable
- Arithmetic saturation (clipping) or wrap-around for MSB truncation
- Round to nearest logic for LSB truncation
- Signed and unsigned data operation

Table 1: Pin Description

Pin Name	Width	Direction	Function
din	*width* bit(s)	Input	Input data
tc	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
sat	1 bit	Input	Saturation enable 0 = no saturation 1 = enable saturation
rnd	1 bit	Input	Rounding enable 0 = no rounding 1 = enable rounding
ov	1 bit	Output	Overflow status
dout	*msb_out –lsb_out +1* bit(s)	Output	Output data

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 2 Default: 16	Word length of din
msb_out	*width*–1 ≥ *msb_out* > *lsb_out* Default: 15	dout MSB position after truncation of din MSBs
lsb_out	*msb_out* > *lsb_out* ≥ 0 Default: 0	dout LSB position after truncation of din LSBs

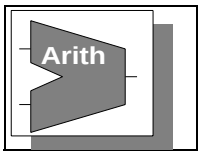


Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_shifter

Combined Arithmetic and Barrel Shifter

- Dynamically selectable arithmetic or barrel shift mode
- Parameterized input control (inverted and non-inverted logic)
- Parameterized padded logic value control (for arithmetic shift only)
- Parameterized data and shift coefficient word lengths
- Inferable using a function call (support for `inv_mode = 0` only)

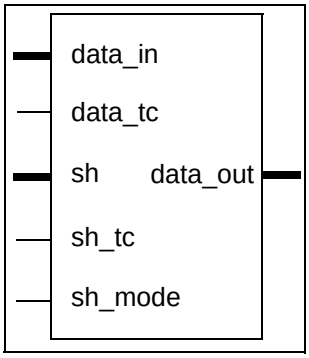


Table 1: Pin Description

Pin Name	Width	Direction	Function
data_in	*data_width* bit(s)	Input	Input data
data_tc	1 bit	Input	Two's complement control on data_in 0 = unsigned data_in 1 = signed data_in
sh	*sh_width* bit(s)	Input	Shift control
sh_tc	1 bit	Input	Two's complement control on sh 0 = unsigned sh 1 = signed sh
sh_mode	1 bit	Input	Arithmetic or barrel shift mode 0 = barrel shift mode 1 = arithmetic shift mode
data_out	*data_width* bit(s)	Output	Output data

Table 2: Parameter Description

Parameter	Values	Description
data_width	≥ 2	Word length of data_in and data_out
sh_width	1 to (ceil($\log_2$[*data_width*]) + 1)	Word length of sh
inv_mode	0 to 3 Default: 0	logic mode 0 = normal input, 0 padding in output; 1 = normal input, 1 padding in output; 2 = inverted input[a],0 padding in output; 3 = inverted input, 1 padding in output

a. Inverted input refers to sh, sh_tc, and data_tc pins only.

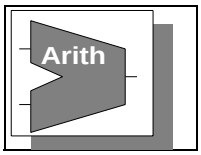


Table 3: Synthesis Implementations

Implementation Name	Function	License Required
mx2	Implement using 2:1 multiplexers only	DesignWare-Foundation
mx2i	Implement using 2:1 inverting multiplexers and 2:1 multiplexers	DesignWare-Foundation
mx4	Implement using 4:1 and 2:1 multiplexers	DesignWare-Foundation
mx8	Implement using 8:1, 4:1, and 2:1 multiplexers	DesignWare-Foundation

DW_square

Integer Squarer

- Parameterized word length
- Unsigned and signed (two's complement) data operation

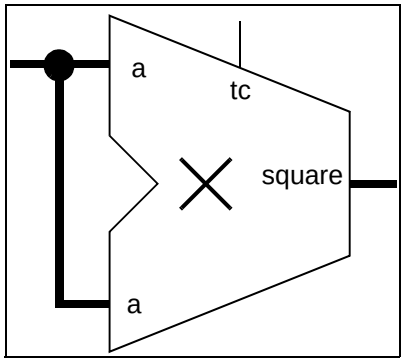


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Input data
tc	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
square	2 × *width* bit(s)	Output	Product of (a × a)

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of a

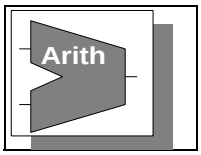
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
wall	Wallace-tree synthesis model	DesignWare-Foundation
mcarch[bc]	MC-inside-DW Wallace-tree	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

b. Automatically chooses Booth-recoding or non-Booth-recoding architecture, depending on constraints.

c. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to 'true'. For more details see the *DesignWare Building Block IP Users Guide*.



DW_squarep

Partial Product Integer Squarer

- Parameterized word lengths
- Unsigned and signed (two's-complement) data operation

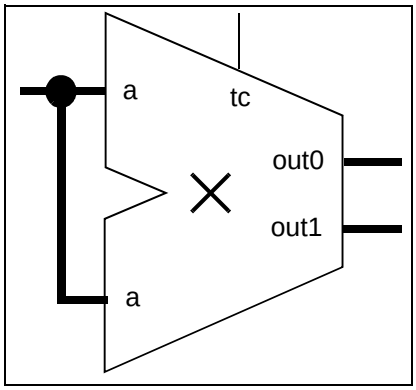


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Multiplier
tc	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
out0	*width* × 2 bit(s)	Output	Partial product of a × a
out1	*width* × 2 bit(s)	Output	Partial product of a × a

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of signal a

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
wall	Wallace-tree synthesis mode	DesignWare-Foundation

DW_sqrt

Combinational Square Root

- Parameterized word length
- Unsigned and signed (two's complement) square root computation

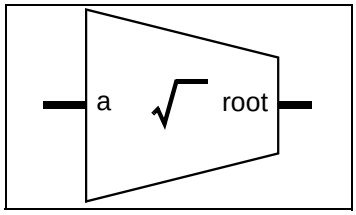


Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*width* bit(s)	Input	Radicand
root	int([*width*+1]/2) bit(s)	Output	Square root

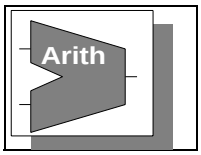
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 2	Word length of a
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = signed

Table 3: Synthesis Implementations[a]

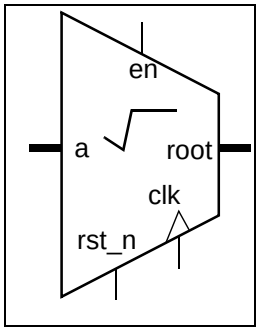
Implementation Name	Function	License Required
rpl	Restoring ripple-carry synthesis model	DesignWare-Foundation
cla	Restoring carry-lookahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_sqrt_pipe

Stallable Pipelined square root



- Parameterized word length
- Unsigned and signed (two's complement) data operation
- Parameterized number of pipeline stages
- Parameterized stall mode (stallable or non-stallable)
- Parameterized reset mode (no reset, asynchronous or synchronous reset)
- Automatic pipeline retiming

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset, active-low (not used if parameter rst_mode=0)
en	1 bit	Input	Load enable (used only if parameter stall_mode=1) 0 = stall 1 = load
a	*width* bit(s)	Input	Radicand
root	(*width*+1)/2 bit(s)	Output	Square root

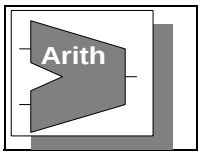
Table 2: Parameter Description

Parameter	Values	Description
width	≥ 2 Default: None	Word length of a
num_stages	≥ 2 Default: 2	Number of pipeline stages
stall_mode	0 or 1 Default: 1	Stall mode 0 = non-stallable 1 = stallable
rst_mode	0 to 2 Default: 1	Reset mode 0 = no reset 1 = asynchronous reset 2 = synchronous reset)

Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str[a]	Pipelined str synthesis model	DesignWare-Foundation

a. One of rpl or cla implementation is selected based the constraints of the design.



DW01_sub
Subtractor

- Parameterized word length
- Carry-in and carry-out signals

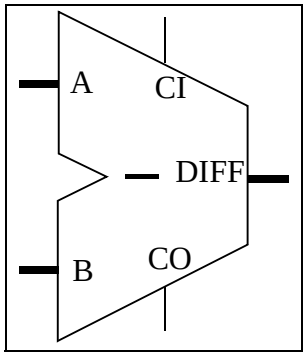


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
CI	1 bit	Input	Carry-in
DIFF	*width* bit(s)	Output	Difference of A –B –CI
CO	1 bit	Output	Carry-out

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of A, B, and DIFF

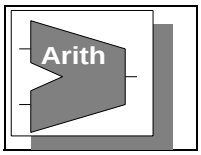
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
bk	Brent-Kung synthesis model	DesignWare-Foundation
clf	Fast carry-look-ahead synthesis model	DesignWare-Foundation
csm[b]	Conditional-sum synthesis model	DesignWare-Foundation
rpcs	Ripple-carry-select synthesis model	DesignWare-Foundation
clsa[c]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation
csa[c]	MC-inside-DW carry-select	DesignWare-Foundation
fastcla[c]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation
pprefix[c]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

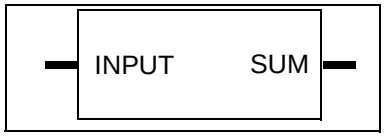
b. The performance of the csm implementation is heavily dependent on the use of a high performance inverting 2-to-1 multiplexer in the technology library. In such libraries, the csm implementation exhibits a superior area-delay product. Although the csm implementation does not always surpass the delay performance of the clf implementation, it is much lower in area.

c. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.



DW02_sum

Vector Adder



- Parameterized number of inputs
- Parameterized word length
- Multiple synthesis implementations

Table 1: Pin Description

Pin Name	Width	Direction	Function
INPUT	*num_inputs* × *input_width* bit(s)	Input	Concatenated input data
SUM	*input_width* bit(s)	Output	Sum

Table 2: Parameter Description

Parameter	Values	Description
num_inputs	≥ 1	Number of inputs
input_width	≥ 1	Word length of inputs and sum

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
csa	Carry-save array synthesis model	DesignWare-Foundation
clsa[b]	MC-inside-DW carry-look-ahead-select	DesignWare-Foundation
fastcla[b]	MC-inside-DW fast carry-look-ahead	DesignWare-Foundation
mccsa[b]	MC-inside-DW carry-select	DesignWare-Foundation
pprefix[b]	MC-inside-DW flexible parallel-prefix	DesignWare-Foundation
ripple[b]	MC-inside-DW ripple-carry	DesignWare-Foundation
rpl	Ripple-carry synthesis model	DesignWare-Foundation
wall	Wallace-tree synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP Users Guide*.

b. This architecture is specially generated using Module Compiler technology. It is normally used as a replacement for, rather than in conjunction with, the HDL architectures available for the same DesignWare part. To use this architecture during synthesis, the dc_shell variable dw_prefer_mc_inside must be set to ‘true’. For more details see the *DesignWare Building Block IP Users Guide*.

DW02_tree

Wallace Tree Compressor

- Parameterized word length

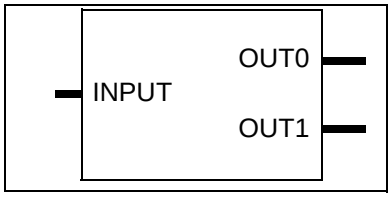


Table 1: Pin Description

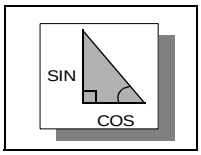
Pin Name	Width	Direction	Function
INPUT	*num_inputs* × *input_width* bit(s)	Input	Input vector
OUT0	*input_width* bit(s)	Output	Partial sum
OUT1	*input_width* bit(s)	Output	Partial sum

Table 2: Parameter Description

Parameter	Values	Description
num_inputs	≥ 1	Number of inputs
input_width	≥ 1	Word length of OUT0 and OUT1

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
wallace	Wallace tree synthesis model	DesignWare-Foundation



Datapath – Trigonometric Overview

The trigonometric IP, many of which are inferred, are applicable to ASIC or FPGA designs. These IP are high performance trigonometric implementations (based on a fast carry look-ahead architecture).

DW02_cos

Combinational Cosine

- Parameterized word length

Table 1: Pin Description

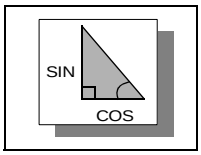
Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Angle in binary
COS	*cos_width* bit(s)	Output	Cosine value of A

Table 2: Parameter Description

Parameter	Values	Description
A_width	2 to 34	Word length of A
cos_width	2 to 34	Word length of COS

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW02_sin

Combinational Sine

- Parameterized word length

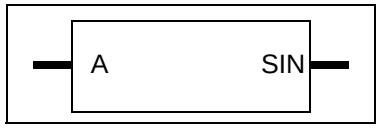


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Angle in binary
SIN	*sin_width* bit(s)	Output	Sine value of A

Table 2: Parameter Description

Parameter	Values	Description
A_width	2 to 34	Word length of A
sin_width	2 to 34	Word length of SIN

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW02_sincos

Combinational Sine - Cosine

- Parameterized word length

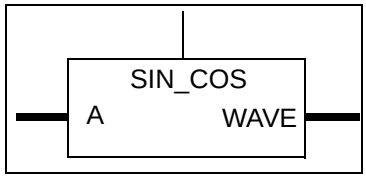


Table 1: Pin Description

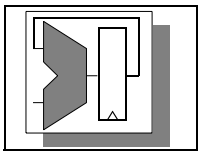
Pin Name	Width	Direction	Function
A	*A_width* bit(s)	Input	Angle in binary
SIN_COS	1 bit	Input	sine (SIN_COS = 0) or cosine (SIN_COS = 1)
WAVE	*wave_width* bit(s)	Output	sine or cosine value of A

Table 2: Parameter Description

Parameter	Values	Function
A_width	2 to 34	Word length of A
wave_width	2 to 34	Word length of WAVE

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



Datapath – Sequential Overview

This section documents the various Datapath - Sequential IP found in the DesignWare Building Block IP.

DW_div_seq

Sequential Divider

- Parameterized word length
- Parameterized number of clock cycles
- Unsigned and signed (two's complement) data multiplication
- Registered or un-registered inputs and outputs

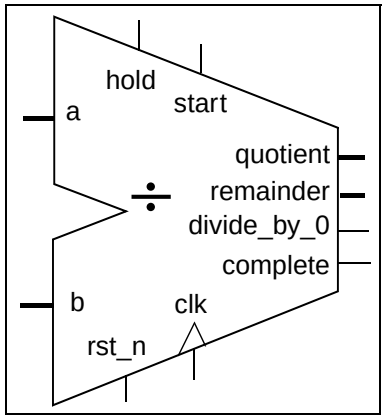


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
hold	1 bit	Input	Hold current operation (=1)
start	1 bit	Input	Start operation (=1). A new operation is started by setting start=1 for one clock cycle.
a	*a_width* bit(s)	Input	Dividend
b	*b_width* bit(s)	Input	Divisor
complete	1 bit	Output	Operation completed (=1)
divide_by_0	1 bit	Output	Indicates if b equals 0
quotient	*a_width* bit(s)	Output	Quotient
remainder	*b_width* bit(s)	Output	Remainder

Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 3	Word length of a
b_width	≥ 3 and ≤*a_width*	Word length of b
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = two's complement
num_cyc	≥ 3 and ≤*a_width* Default: 3	User-defined number of clock cycles to produce a valid result. The real number of clock cycles depends on various parameters.

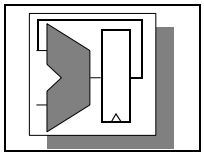


Table 2: Parameter Description (Continued)

Parameter	Values	Description
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset 1 = synchronous reset
input_mode	0 or 1 Default: 1	Registered inputs 0 = no 1 = yes
output_mode	0 or 1 Default: 1	Registered outputs 0 = no 1 = yes
early_start	0 or 1 Default: 0	Computation start 0 = start computation in the second cycle 1 = start computation in the first cycle

Table 3: Synthesis Implementations

Implementation	Function	License Required
cpa	Carry-propagate adder synthesis model	DesignWare-Foundation

DW_mult_seq

Sequential Multiplier

- Parameterized word length
- Parameterized number of clock cycles
- Unsigned and signed (two's complement) data multiplication
- Registered or un-registered inputs and outputs.

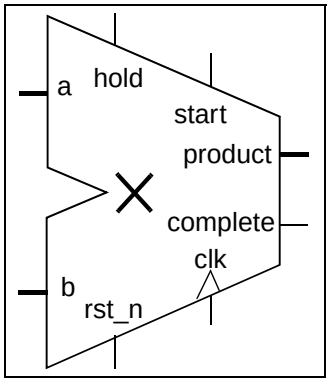


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
hold	1 bit	Input	Hold current operation (=1)
start	1 bit	Input	Start operation (=1). A new operation is started again by making start=1 for one clock cycle.
a	*a_width* bit(s)	Input	Multiplier
b	*b_width* bit(s)	Input	Multiplicand
complete	1 bit	Output	Operation completed (=1)
product	*a_width* + *b_width* bit(s)	Output	Product a × b

Table 2: Parameter Description

Parameter	Values	Description
a_width	≥ 3 and ≤*b_width*	Word length of a
b_width	≥ 3	Word length of b
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = two's complement
num_cyc	≥ 3 and ≤*a_width* Default: 3	User-defined number of clock cycles to produce a valid result. The real number of clock cycles depends on various parameters.
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset 1 = synchronous reset

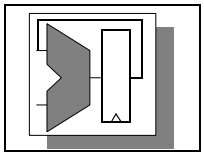


Table 2: Parameter Description (Continued)

Parameter	Values	Description
input_mode	0 or 1 Default: 1	Registered inputs 0 = no 1 = yes
output_mode	0 or 1 Default: 1	Registered outputs 0 = no 1 = yes
early_start	0 or 1 Default: 0	Computation start 0 = start computation in the second cycle 1 = start computation in the first cycle

Table 3: Synthesis Implementations

Implementation	Function	License Required
cpa	Carry-propagate adder synthesis model	DesignWare-Foundation

DW_sqrt_seq
Sequential Square Root

- Parameterized word length
- Parameterized number of clock cycles
- Unsigned and signed (two's complement) data multiplication
- Registered or un-registered inputs and outputs

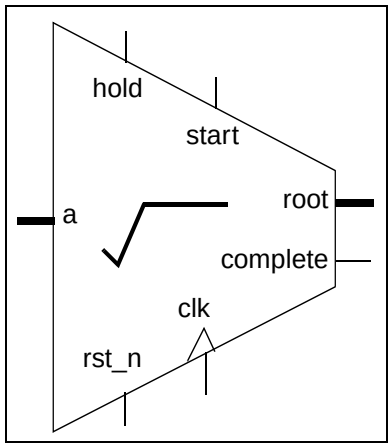


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
hold	1 bit	Input	Hold current operation (=1)
start	1 bit	Input	Start operation (=1). A new operation is started by setting start=1 for one clock cycle.
a	*width* bit(s)	Input	Radicand
complete	1 bit	Output	Operation completed (=1)
root	*(width* +1)/2 bit(s)	Output	Square root

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 6	Word length of a
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = two's complement
num_cyc	≥ 3 and ≤*width* Default: 3	User-defined number of clock cycles to produce a valid result. The real number of clock cycles depends on various parameters.
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset 1 = synchronous reset

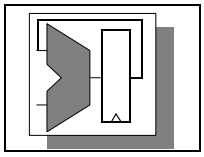


Table 2: Parameter Description (Continued)

Parameter	Values	Description
input_mode	0 or 1 Default: 1	Registered inputs 0 = no 1 = yes
output_mode	0 or 1 Default: 1	Registered outputs 0 = no 1 = yes
early_start	0 or 1 Default: 0	Computation start 0 = start computation in the second cycle 1 = start computation in the first cycle

Table 3: Synthesis Implementations

Implementation	Function	License Required
cpa	Carry-propagate adder synthesis model	DesignWare-Foundation

Datapath – Floating Point Overview

Note

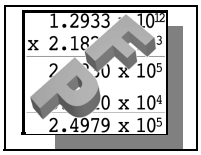
The Floating Point IP are designed specifically for Module Compiler and do not work with Design Compiler.

The Floating Point IP are included in the DesignWare Building Block IP set as a convenience. You can find the Floating Point DesignWare Building Block IP group download request instructions at the following web address:

http://www.synopsys.com/products/designware/dwest/dwfloating_form.html

The Floating Point DesignWare Building Block IP group is a library of functions used to synthesize floating point computational circuits in high end ASICs. The functions mainly deal with arithmetic operations in floating point format, format conversions and comparison functions. The main features of this library are as follows:

- The format of the floating point numbers that determines the precision of the number that it represents is parametrizable. The user can select the precision based on either IEEE single or double precision, or custom format defined by you.
- The parameter range for exponents is from 3 to 31 bits.
- The parameter range for the significand or the fractional part of the floating point number is from 2 bits to 256 bits.
- The parameter range for integers is from 3 to 512 bits.
- Accuracy conforms to the definitions in the IEEE 754 Floating Point standard.



DW_add_fp

Floating Point Adder - (Module Compiler Only)

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand or fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard

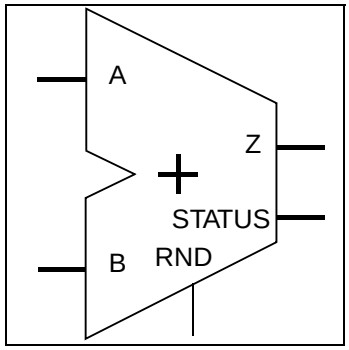


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	e+f+1 bits	Input	Input data
B	e+f+1 bits	Input	Input data
Z	e+f+1 bits	Output	Sum of A + B
STATUS (optional)	8 bits	Output	Status flags
RND (optional)	3 bits	Input	Rounding mode

Table 2: Parameter Description

Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent of floating point numbers A, B, and Z
f	2 to 253 bits	Word length of fraction field of floating point numbers A, B, and Z
arch[a]	0	Architecture implementation

a. The DW_add_fp component contains only one architecture, therefore the arch parameter should be set to 0.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation

DW_cmp_fp

Floating Point Comparator - (Module Compiler Only)

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand or fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard

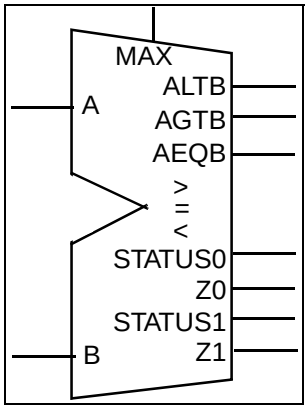


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	e+f+1 bits	Input	Floating point number
B	e+f+1 bits	Input	Floating point number
ALTB	1 bit	Output	High when A is less than B
AGTB	1 bit	Output	High when A is greater than B
AEQB	1 bit	Output	High when A is equal to B
Z0	e+f+1 bits	Output	Optional floating point output of e+f+1 bits
Z1	e+f+1 bits	Output	Optional floating point output of e+f+1 bits
STATUS0 (optional)	8 bits	Output	Status flags corresponding to Z0
STATUS1 (optional)	8 bits	Output	Status flags corresponding to Z1
MAX (optional)	1 bit	Input	Determines Min/Max operation of Z0 and Z1

Table 2: Parameter Description

Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent field of floating point number A
f	2 to 256 bits	Word length of fraction field of floating point number A
arch[a]	0	Architecture implementation

a. This component contains only one architecture. Therefore, the arch parameter should be set to 0.

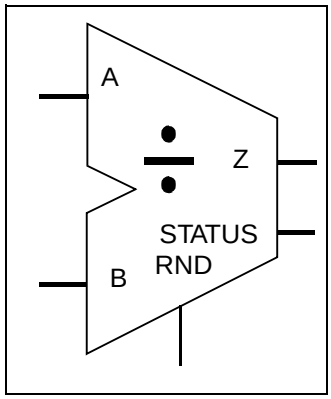
Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation

DW_div_fp

Floating Point Divider - (Module Compiler Only)

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand and fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard



Note

The Floating Point Divider is designed specially for Module Compiler and does not work with Design Compiler.

Table 1: Pin Description

Pin Name	Width	Direction	Function
A	e+f+1 bits	Input	Dividend
B	e+f+1 bits	Input	Divisor
Z	e+f+1 bits	Output	Quotient of A/B
STATUS (optional)	8 bits	Output	Status flags
RND (optional)	3 bits	Input	Rounding mode

Table 2: Parameter Description

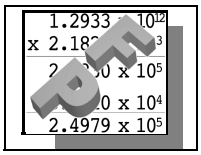
Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent of floating point numbers A, B, and Z
f	2 to 253 bits	Word length of fraction field of floating point numbers A, B, and Z

Table 2: Parameter Description (Continued)

Parameter	Values	Description
arch	1, 2, and 3	Architecture implementation 1: MC_divider architecture 1, producing 1-bit per iteration 2: MC_divider ROM based architecture, producing 1-bit per iteration 3: MC_divider architecture 3, producing 2-bits per iteration Divider operands can be of any width for arch=1 and arch=3, but should be less than 10 bits for arch=2. For details, see divide() function of Module Compiler reference manual.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation



DW_flt2i_fp

Floating Point-to-Integer Converter - (Module Compiler Only)

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand or fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard

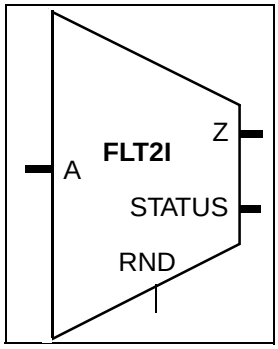


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	e+f+1 bits	Input	Floating point number
Z	3 to 512 bits	Output	Two’s complement integer number
STATUS (optional)	8 bits	Output	Status flags
RND (optional)	3 bits	Input	Rounding mode

Table 2: Parameter Description

Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent of floating point number A
f	2 to 256 bits	Word length of fraction field of floating point number A
arch[a]	0	Architecture implementation

a. This component contains only one architecture. Therefore, the arch parameter should be set to 0.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation

DW_i2flt_fp

Integer-to-Floating Point Converter - (Module Compiler Only)

- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand or fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard

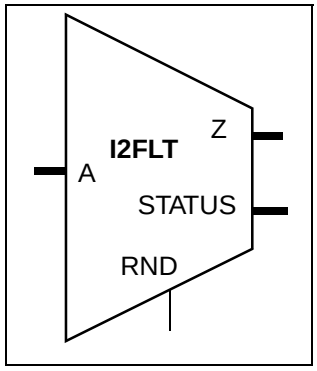


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	3 to 512 bits	Input	Two's compliment integer number
Z	e+f+1 bits	Output	Floating point number
STATUS (optional)	8 bits	Output	Status flags
RND (optional)	3 bits	Input	Rounding mode

Table 2: Parameter Description

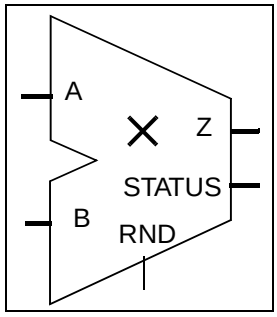
Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent of floating point number A
f	2 to 256 bits	Word length of fraction field of floating point number A
arch	0	Architecture implementation

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation

DW_mult_fp

Floating Point Multiplier - (Module Compiler Only)



- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Exponents can range from 3 to 31 bits
- Significand or fractional part of the floating point number can range from 2 to 256 bits
- Accuracy conforms to IEEE 754 Floating Point standard

Table 1: Pin Description

Pin Name	Width	Direction	Function
A	e+f+1 bits	Input	Multiplier
B	e+f+1 bits	Input	Multiplicand
Z	e+f+1 bits	Output	Product of A × B
STATUS (optional)	8 bits	Output	Status flags
RND (optional)	3 bits	Input	Rounding mode

Table 2: Parameter Description

Parameter	Values	Description
e	3 to 31 bits	Word length of biased exponent of floating point numbers A, B, and Z
f	2 to 253 bits	Word length of fraction field of floating point numbers A, B, and Z
arch[a]	0	Architecture implementation

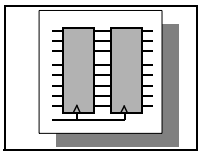
a. This component contains only one architecture, therfore the arch parameter should be set to 0.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
arch0	Synthesis model	DesignWare-Foundation

Memory – Registers

This section documents the various memory registers found in the library of DesignWare Building Block IP.



DW03_pipe_reg

Pipeline Register

- Parameterized data width and depth

Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data bus
clk	1 bit	Input	Clock
B	*width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
depth	≥ 1	Depth of registers
width	≥ 1	Width of A and B buses

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW03_reg_s_pl

Register with Synchronous Enable Reset

- Parameterizable data width
- Parameterized reset to any constant value
- Multiple synthesis implementations

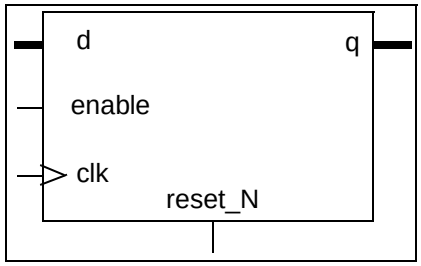


Table 1: Pin Description

Pin Name	Width	Direction	Function
d	*width* bit(s)	Input	Input data bus
clk	1 bit	Input	Clock
reset_N	1 bit	Input	Synchronous reset
enable	1 bit	Input	Enables all operations
q	*width* bi(s)	Output	Output data bus

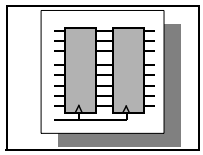
Table 2: Parameter Description

Parameter	Values	Description
width	1 to 31 Default: 8	Width of d and q buses
reset_value	0 to $2^{width}-1$ when width ≤31; 0 when width ≥ 32 Default: 0	Resets to a constant

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
str	Single-bit flip-flops synthesis model	DesignWare-Components
mbstr	Multiple-bit flip-flops synthesis model	DesignWare-Components

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW03_shftreg

Shift Register

- Parameterized word length
- Active low shift enable
- Active low load enable

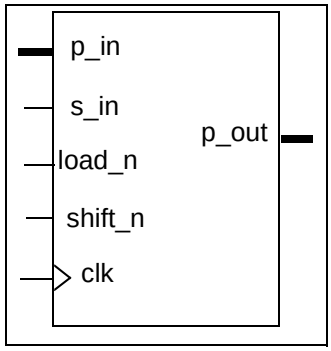


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
s_in	1 bit	Input	Serial shift input
p_in	*length* bit(s)	Input	Parallel input
shift_n	1 bit	Input	Shift enable, active low
load_n	1 bit	Input	Parallel load enable, active low
p_out	*length* bit(s)	Output	Shift register parallel output

Table 2: Parameter Description

Parameter	Values	Description
length	≥ 1	Length of shifter

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW04_shad_reg

Shadow and Multibit Register

- Captures the state of system registers dynamically during system operation
- Serial access on shadow register to scan out the state of captured data
- Constructed with multibit flip-flop cells where possible; can be used as a simple, non-shadowed multibit register
- Parameterized width and number of registers (one or two)

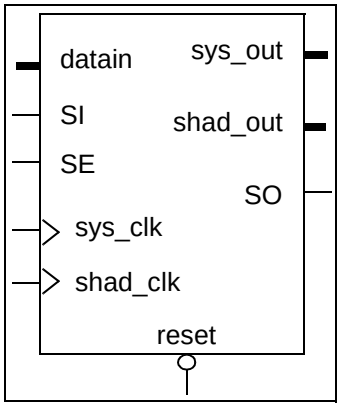


Table 1: Pin Description

Pin Name	Width	Direction	Function
datain	*width* bit(s)	Input	Data input driving the input to the system register
sys_clk	1 bit	Input	Clock that samples the system register, positive edge triggered
shad_clk	1 bit	Input	Signal that clocks the output of the system register into the shadow register, positive edge triggered
reset	1 bit	Input	Asynchronous reset signal that clears the system and shadow registers
SI	1 bit	Input	Serial scan input, clocked by shad_clk when SE is high
SE	1 bit	Input	Serial scan enable signal, active high. Enables scan only on the shadow register.
sys_out	*width* bit(s)	Output	Output of the system register
shad_out	*width* bit(s)	Output	Parallel output of the shadow register that lags the system register by one cycle
SO	1 bit	Output	Serial scan output from shadow register. When SE is low, represents the state of the MSB of the shadow register. When SE is high, each successive bit is shifted up one and SI is clocked into the LSB.

Table 2: Parameter Description

Parameter	Values	Description
width[1]	1 to 512[a]	Defines the width of the system and shadow registers, and the input and output buses
bld_shad_reg	0 or 1	Defines whether to build both the system and shadow registers (bld_shad_reg = 1) or just the system register (bld_shad_reg = 0)

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

DW04_shad_reg
Shadow and Multibit Register

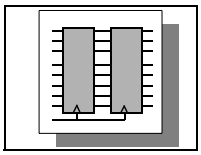


Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

Memory – FIFO Overview

The FIFOs in this category address a broad array of design requirements. FIFOs, which include dual-port RAM memory arrays, are offered for both synchronous and asynchronous interfaces. The memory arrays are offered in two configurations: latch-based to minimize area, and D flip-flop-based to maximize testability. These two configurations also offer flexibility when working under design constraints, such as a requirement that no latches be employed. Flip-flop-based designs employ no clock gating to minimize skew and maximize performance. All FIFOs employ a FIFO RAM controller architecture in which there is no extended “fall-through” time required before reading contents just written.

Also offered are FIFO Controllers without the RAM array. They consist of control and flag logic and an interface to common ASIC dual port RAMs. Choosing between the two is typically based on the required size of the FIFO. For shallow FIFOs (less than 256 bits), synchronous or asynchronous FIFOs are available which include both memory and control in a single macro. These macros can be programmed via word width, depth, and level (almost-full flag) parameters.

For larger applications (greater than 256 bits), you can use the asynchronous FIFO Controller with a diffused or metal programmable RAM. See Figure 1.

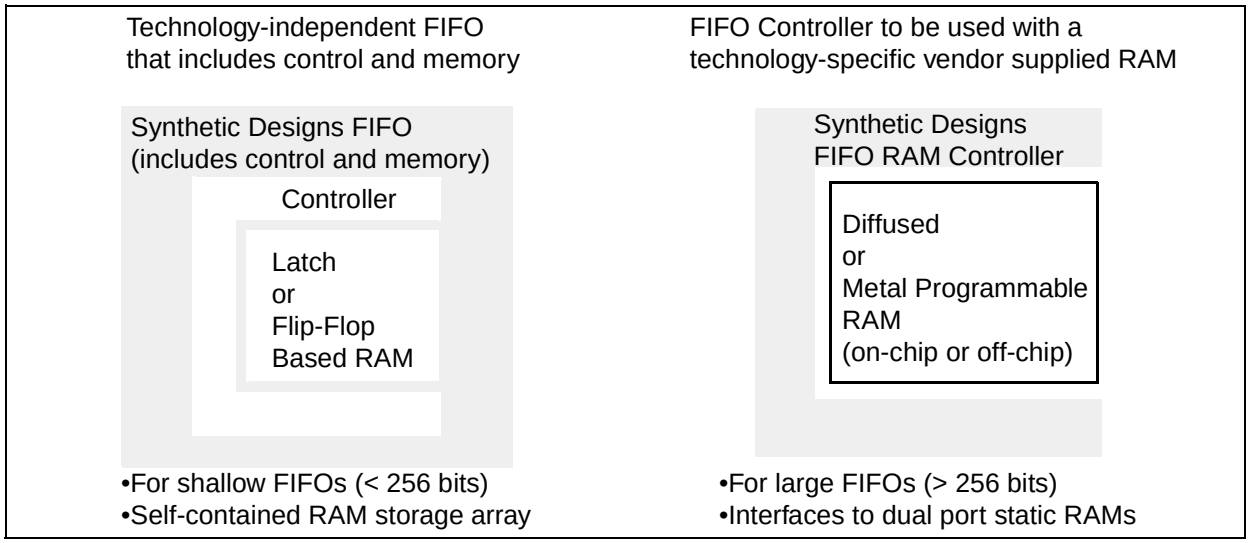
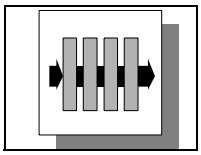


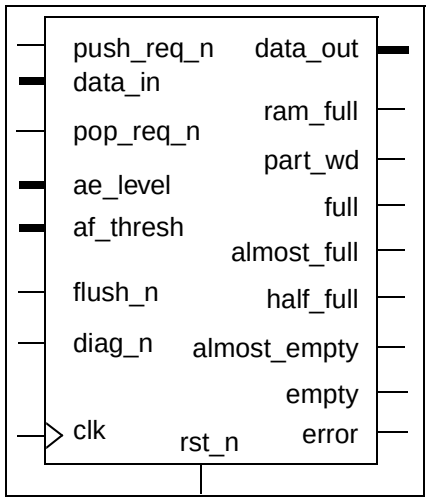
Figure 1: Memory: FIFOs and FIFO Controllers

All FIFOs and Controllers support full, empty, and programmable flag logic. Programmable flag logic may be statically or dynamically programmed. When statically programmed, the threshold comparison value is hardwired at synthesis compile time. When dynamically programmed, it may be changed during FIFO operation.



DW_asymfifo_s1_df

Asymmetric I/O Synchronous (Single Clock) FIFO with Dynamic Flag



- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- Parameterized asymmetric input and output bit widths (must be integer-multiple relationship)
- Word integrity flag for *data_in_width* < *data_out_width*
- Flushing out partial word for *data_in_width* < *data_out_width*
- Parameterized byte (or subword) order within a word
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Dynamically programmable almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low (asynchronous if rst_mode = 0, synchronous if rst_mode = 1)
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's) (for *data_in_width* < *data_out_width* only)
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low (for err_mode = 0, NC for other err_mode values)
data_in	*data_in_width* bit(s)	Input	FIFO data to push
ae_level	ceil($\log_2$[*depth*]) bit(s)	Input	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_thresh	ceil($\log_2$[*depth*]) bit(s)	Input	Almost full threshold (the number of words stored in the FIFO at or above which the almost_full flag is active)
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high, asserted when FIFO level ≤*ae_level*
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high, asserted when FIFO level ≥ (*af_thresh*)
full	1 bit	Output	FIFO full output, active high
ram_full	1 bit	Output	RAM full output, active high
error	1 bit	Output	FIFO error output, active high
part_wd	1 bit	Output	Partial word, active high (for *data_in_width* < *data_out_width* only; otherwise, tied low)
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

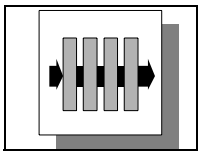


Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. data_in_width must be in an integer-multiple relationship with data_out_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be in an integer-multiple relationship with data_in_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
depth	2 to 256	Number of memory elements used in the FIFO (addr_width = ceil[$\log_2$(*depth*)])
err_mode	0 to 2 Default: 1	Error mode 0 = underflow/overflow with pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking.
rst_mode	0 to 3 Default: 1	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory.
byte_order	0 or 1 Default: 0	Order of send/receive bytes or subword [subword - 8 bits - subword] within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position [valid for *data_in_width* ≠ *data_out_width*]).

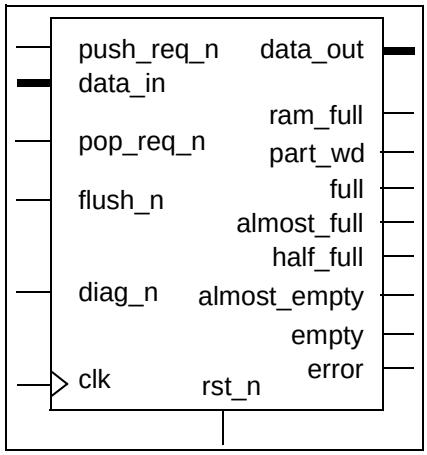
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_asymfifo_s1_sf

Asymmetric I/O Synchronous (Single Clock) FIFO with Static Flags



- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- Parameterized asymmetric input and output bit widths (must be integer-multiple relationship)
- Word integrity flag for *data_in_width* < *data_out_width*
- Flushing out partial word for *data_in_width* < *data_out_width*
- Parameterized byte (or subword) order within a word
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

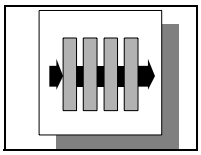


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0, synchronous if rst_mode = 1
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's) (for *data_in_width* < *data_out_width* only)
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low (for err_mode = 0, NC for other err_mode values)
data_in	*data_in_width* bit(s)	Input	FIFO data to push
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high, asserted when FIFO level ≤*ae_level*
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high, asserted when FIFO level ≥ (*depth – af_level*)
full	1 bit	Output	FIFO full output, active high
ram_full	1 bit	Output	RAM full output, active high
error	1 bit	Output	FIFO error output, active high
part_wd	1 bit	Output	Partial word, active high (for *data_in_width* < *data_out_width* only; otherwise, tied low)
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. data_in_width must be in an integer-multiple relationship with data_out_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be in an integer-multiple relationship with data_in_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
depth	2 to 256	Number of memory elements used in the FIFO (addr_width = ceil[$\log_2$(*depth*)])
ae_level	1 to *depth* – 1	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_level	1 to *depth* – 1	Almost full level (the number of empty memory locations in the FIFO at which the almost_full flag is active.
err_mode	0 to 2 Default: 1	Error mode 0 = underflow/overflow with pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking).
rst_mode	0 to 3 Default: 1	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory).
byte_order	0 or 1 Default: 0	Order of send/receive bytes or subword [subword < 8 bits > subword] within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position [valid for *data_in_width* ≠ *data_out_width*]).

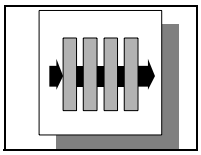


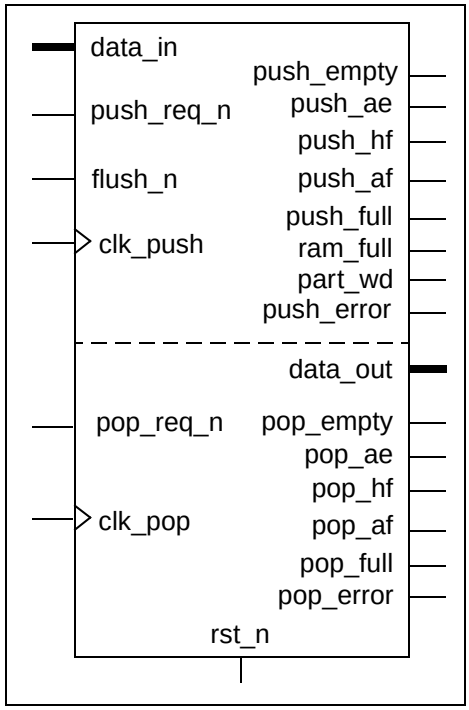
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_asymfifo_s2_sf

Asymmetric Synchronous (Dual Clock) FIFO with Static Flags



- Parameterized asymmetric input and output bit widths (must be integer-multiple relationship)
- Fully registered synchronous flag output ports
- Separate status flags for each clock system
- FIFO empty, half full, and full flags
- Parameterized almost full and almost empty flags
- FIFO push error (overflow) and pop error (underflow) flags
- D flip-flop-based memory array for high testability
- Single clock cycle push and pop operations
- Word integrity flag for *data_in_width* < *data_out_width*
- Partial word flush for *data_in_width* < *data_out_width*
- Parameterized byte order within a word
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk_push	1 bit	Input	Input clock for push interface
clk_pop	1 bit	Input	Input clock for pop interface
rst_n	1 bit	Input	Reset input, active low
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's for empty bits) (for *data_in_width* < *data_out_width* only), active low
pop_req_n	1 bit	Input	FIFO pop request, active low
data_in	*data_in_width* bit(s)	Input	FIFO data to push

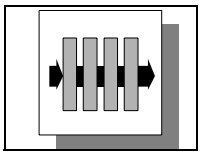


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
push_empty	1 bit	Output	FIFO empty[a] output flag synchronous to clk_push, active high
push_ae	1 bit	Output	FIFO almost empty[a] output flag synchronous to clk_push (determined by push_ae_lvl parameter), active high
push_hf	1 bit	Output	FIFO half full[a] output flag synchronous to clk_push, active high
push_af	1 bit	Output	FIFO almost full[a] output flag synchronous to clk_push (determined by push_af_lvl parameter), active high
push_full	1 bit	Output	FIFO's RAM full[a] output flag (including the input buffer of FIFO for *data_in_width* < *data_out_width*) synchronous to clk_push, active high
ram_full	1 bit	Output	FIFO's RAM (excluding the input buffer of FIFO for *data_in_width* < *data_out_width*) full output flag synchronous to clk_push, active high
part_wd	1 bit	Output	Partial word accumulated in the input buffer synchronous to clk_push (for *data_in_width* < *data_out_width* only; otherwise, tied low), active high
push_error	1 bit	Output	FIFO push error (overrun) output flag synchronous to clk_push, active high
pop_empty	1 bit	Output	FIFO empty[b] output flag synchronous to clk_pop, active high
pop_ae	1 bit	Output	FIFO almost empty[b] output flag synchronous to clk_pop (determined by pop_ae_lvl parameter), active high
pop_hf	1 bit	Output	FIFO half full[b] output flag synchronous to clk_pop, active high
pop_af	1 bit	Output	FIFO almost full[b] output flag synchronous to clk_pop (determined by pop_af_lvl parameter), active high
pop_full	1 bit	Output	FIFO's RAM full[b] output flag (excluding the input buffer of FIFO for case *data_in_width* < *data_out_width*) synchronous to clk_pop, active high
pop_error	1 bit	Output	FIFO pop error (underrun) output flag synchronous to clk_pop, active high
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

a. As perceived by the push interface.

b. As perceived by the pop interface.

Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. data_in_width must in an integer-multiple of data_out_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be an integer-multiple of data_in_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
depth	4 to 256	Number of words that can be stored in FIFO
push_ae_lvl	1 to *depth*-1	Almost empty level for the push_ae output port (the number of words in the FIFO at or below which the push_ae flag is active).
push_af_lvl	1 to *depth*-1	Almost full level for the push_af output port (the number of empty memory locations in the FIFO at which the push_af flag is active.)
pop_ae_lvl	1 to *depth*-1	Almost empty level for the pop_ae output port (the number of words in the FIFO at or below which the pop_ae flag is active)
pop_af_lvl	1 to *depth*-1	Almost full level for the pop_af output port (the number of empty memory locations in the FIFO at which the pop_af flag is active.)
err_mode	0 or 1	Error mode 0 = stays active until reset [latched], 1 = active only as long as error condition exists [unlatched]
push_sync	1 to 3	Push flag synchronization mode 1 = single register synchronization from pop pointer, 2 = double register, 3 = triple register
pop_sync	1 to 3	Pop flag synchronization mode 1 = single register synchronization from push pointer, 2 = double register, 3 = triple register)
rst_mode	0 or 3 Default: 1	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory).

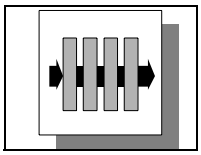


Table 2: Parameter Description (Continued)

Parameter	Values	Description
byte_order	0 or 1 Default: 0	Order of bytes or subword within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position.

a. Valid depth values include binary numbers from 8 to 256 (i.e. 8, 16, 32, 64, etc.) and all odd values between 8 and 256.

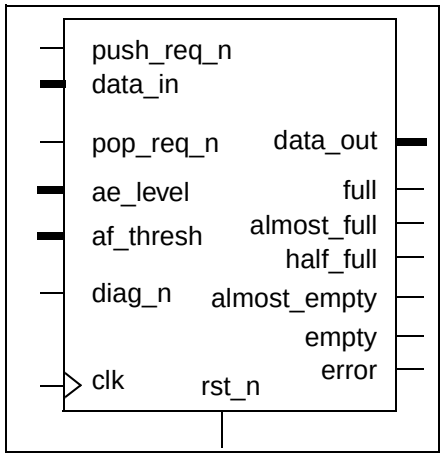
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_fifo_s1_df

Synchronous (Single Clock) FIFO with Dynamic Flags



- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Dynamically programmable almost full and almost empty flags
- Parameterized word width
- Parameterized word depth
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low (asynchronous if rst_mode = 0 or 2, synchronous if rst_mode = 1 or 3)
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low
ae_level	ceil($\log_2$[*depth*]) bit(s)	Input	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_thresh	ceil($\log_2$[*depth*]) bit(s)	Input	Almost full threshold (the number of words stored in the FIFO at or above which the almost_full flag is active)
data_in	*width* bit(s)	Input	FIFO data to push
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high
half_full	1 bit	Output	FIFO half full output, active high

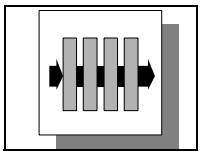


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
almost_full	1 bit	Output	FIFO almost full output, active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high
data_out	*width* bit(s)	Output	FIFO data to pop

Table 2: Parameter Description

Parameter	Values	Description
width	1 to 256 Default: 8	Width of data_in and data_out buses
depth	2 to 256 Default: 4	Number of memory elements used in FIFO
err_mode	0 to 2 Default: 0	Error mode 0 = underflow/overflow and pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking
rst_mode	0 to 3 Default: 0	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory

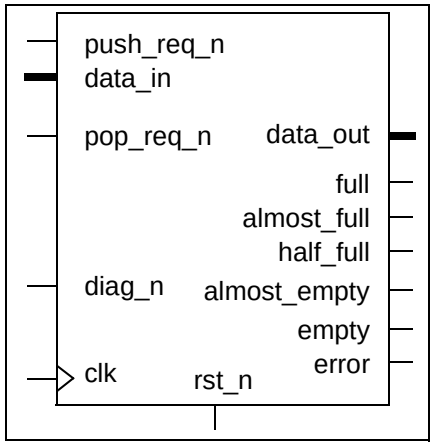
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_fifo_s1_sf

Synchronous (Single Clock) FIFO with Static Flags



- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word width
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0 or 2, synchronous if rst_mode = 1 or 3
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low
data_in	*width* bit(s)	Input	FIFO data to push
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high
data_out	*width* bit(s)	Output	FIFO data to pop

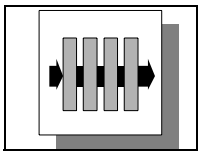


Table 2: Parameter Description

Parameter	Values	Function
width	1 to 256 Default: 8	Width of the data_in and data_out buses
depth	2 to 256 Default: 4	Number of memory elements used in FIFO (addr_width = ceil($\log_2$(*depth*))
ae_level	1 to *depth* – 1 Default: 1	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_level	1 to *depth* – 1 Default: 1	Almost full level (the number of empty memory locations in the FIFO at which the almost_full flag is active.)
err_mode	0 to 2 Default: 0	Error mode 0 = underflow/overflow and pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking
rst_mode	0 to 3 Default: 0	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_fifo_s2_sf

Synchronous (Dual-Clock) FIFO with Static Flags

- Fully registered synchronous flag output ports
- Single clock cycle push and pop operations
- Parameterized word width
- Parameterized word depth
- Separate status flags for each clock system
- FIFO empty, half full, and full flags
- Parameterized almost full and almost empty flag thresholds
- FIFO push error (overflow) and pop error (underflow) flags

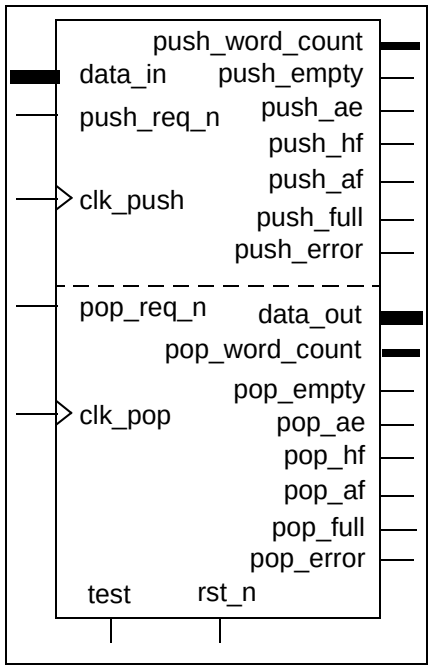


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk_push	1 bit	Input	Input clock for push interface
clk_pop	1 bit	Input	Input clock for pop interface
rst_n	1 bit	Input	Reset input, active low
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
data_in	*width* bit(s)	Input	FIFO data to push
push_empty	1 bit	Output	FIFO empty [a] output flag synchronous to clk_push, active high
push_ae	1 bit	Output	FIFO almost empty[a] output flag synchronous to clk_push, active high (determined by push_ae_lvl parameter)
push_hf	1 bit	Output	FIFO half full[a] output flag synchronous to clk_push, active high
push_af	1 bit	Output	FIFO almost full[a] output flag synchronous to clk_push, active high (determined by push_af_lvl parameter)
push_full	1 bit	Output	FIFO full[a] output flag synchronous to clk_push, active high
push_error	1 bit	Output	FIFO push error (overrun) output flag synchronous to clk_push, active high

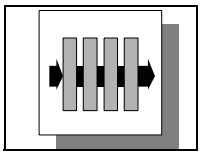


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
pop_empty	1 bit	Output	FIFO empty [b] output flag synchronous to clk_pop, active high
pop_ae	1 bit	Output	FIFO almost empty[b] output flag synchronous to clk_pop, active high (determined by pop_ae_lvl parameter)
pop_hf	1 bit	Output	FIFO half full[b] output flag synchronous to clk_pop, active high
pop_af	1 bit	Output	FIFO almost full[b] output flag synchronous to clk_pop, active high (determined by pop_af_lvl parameter)
pop_full	1 bit	Output	FIFO full[b] output flag synchronous to clk_pop, active high
pop_error	1 bit	Output	FIFO pop error (underrun) output flag synchronous to clk_pop, active high
data_out	*width* bit(s)	Output	FIFO data to pop

a. As perceived by the push interface.

b. As perceived by the pop interface.

Table 2: Parameter Description

Parameter	Values	Description
width	1 to 256 Default: 8	Width of the data_in and data_out buses
depth	4 to 256 Default: 8	Number of words that can be stored in FIFO
push_ae_lvl	1 to *depth*–1 Default: 2	Almost empty level for the push_ae output port (the number of words in the FIFO at or below which the push_ae flag is active)
push_af_lvl	1 to *depth*–1 Default: 2	Almost full level for the push_af output port (the number of empty memory locations in the FIFO at which the push_af flag is active.)
pop_ae_lvl	1 to *depth*–1 Default: 2	Almost empty level for the pop_ae output port (the number of words in the FIFO at or below which the pop_ae flag is active)
pop_af_lvl	1 to *depth*–1 Default: 2	Almost full level for the pop_af output port (the number of empty memory locations in the FIFO at which the pop_af flag is active.)

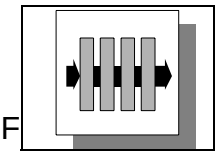
Table 2: Parameter Description (Continued)

Parameter	Values	Description
err_mode	0 or 1 Default: 0	Error mode 0 = stays active until reset [latched], 1 = active only as long as error condition exists [unlatched])
push_sync	1 to 3 Default: 2	Push flag synchronization mode 1 = single register synchronization from pop pointer, 2 = double register, 3 = triple register
pop_sync	1 to 3 Default: 2	Pop flag synchronization mode 1 = single register synchronization from push pointer, 2 = double register, 3 = triple register)
rst_mode	0 to 3 Default: 0	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory

Table 3: Synthesis Implementations[a]

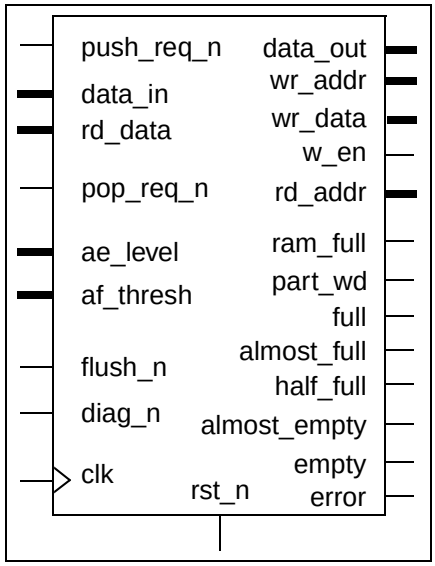
Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_asymfifoctl_s1_df

Asymmetric I/O Synchronous (Single Clock) FIFO Controller with Dynamic Flags



- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- Asymmetric input and output bit widths (must be integer-multiple relationship)
- Word integrity flag for *data_in_width* < *data_out_width*
- Flushing out partial word for *data_in_width* < *data_out_width*
- Parameterized byte order within a word
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Dynamically programmable almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode=0, synchronous if rst_mode=1)
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's) (for *data_in_width* < *data_out_width* only)
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low (for err_mode=0, NC for other err_mode values)
data_in	*data_in_width* bit(s)	Input	FIFO data to push

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
rd_data	max (*data_in_width*, *data_out_width*) bit(s)	Input	RAM data input to FIFO controller
ae_level	ceil($\log_2$[*depth*]) bit(s)	Input	Almost empty level (the number of words in the FIFO at or below which the almost _empty flag is active)
af_thresh	ceil($\log_2$[*depth*]) bit(s)	Input	Almost full threshold(the number of words stored in the FIFO at or above which the almost_full flag is active)
w_en	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high, asserted when FIFO level ≤*ae_level*
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high, asserted when FIFO level ≥ *af_thresh*
full	1 bit	Output	FIFO full output, active high
ram_full	1 bit	Output	RAM full output, active high
error	1 bit	Output	FIFO error output, active high
part_wd	1 bit	Output	Partial word, active high (for *data_in_width* < *data_out_width* only; otherwise, tied low)
wr_data	max (*data_in_width*, *data_out_width*) bit(s)	Output	FIFO controller output data to RAM
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

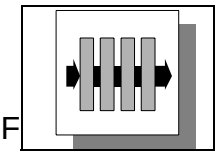


Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. data_in_width must be in an integer-multiple relationship with data_out_width. That is, either *data_in_width* = $K \times$ *data_out_width*, or *data_out_width* = $K \times$ *data_in_width*, where K is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be in an integer-multiple relationship with data_in_width. That is, either *data_in_width* = $K \times$ *data_out_width*, or *data_out_width* = $K \times$ *data_in_width*, where K is an integer.
depth	2 to 2^{24}	Number of memory elements used in the FIFO (*addr_width* = ceil[$\log_2$(*depth*)])
err_mode	0 to 2 Default: 1	Error mode 0 = underflow/overflow with pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking).
rst_mode	0 or 1 Default: 1	Reset mode 0 = asynchronous reset, 1 = synchronous reset).
byte_order	0 or 1 Default: 0	Order of bytes or subword [*subword* < 8 bits > *subword*] within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position.

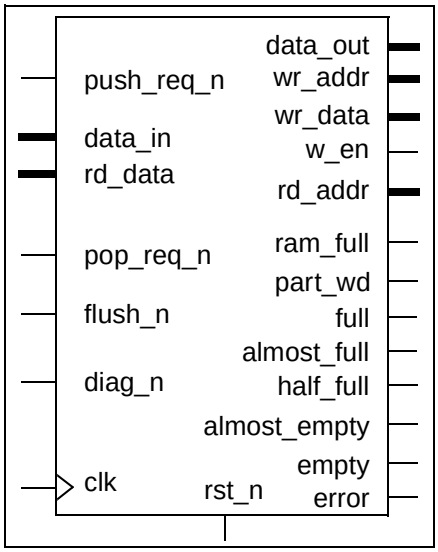
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_asymfifoctl_s1_sf

Asymmetric I/O Synchronous (Single Clock) FIFO Controller with Static Flags



- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- Asymmetric input and output bit widths (must be integer-multiple relationship)
- Word integrity flag for *data_in_width* < *data_out_width*
- Flushing out partial word for *data_in_width* < *data_out_width*
- Parameterized byte order within a word
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0, synchronous if rst_mode = 1)
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's) (for *data_in_width* < *data_out_width* only)
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low (for err_mode = 0, NC for other err_mode values)
data_in	*data_in_width* bit(s)	Input	FIFO data to push

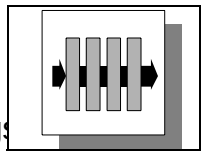


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
rd_data	max (*data_in_width*, *data_out_width*) bit(s)	Input	RAM data input to FIFO controller
w_en	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high, asserted when FIFO level ≤*ae_level*
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high, asserted when FIFO level ≥ (*depth* – *af_level*)
full	1 bit	Output	FIFO full output, active high
ram_full	1 bit	Output	RAM full output, active high
error	1 bit	Output	FIFO error output, active high
part_wd	1 bit	Output	Partial word, active high (for *data_in_width* < *data_out_width* only; otherwise, tied low)
wr_data	max (*data_in_width*, *data_out_width*) bit(s)	Output	FIFO controller output data to RAM
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. Values for data_in_width must be in an integer-multiple relationship with data_out_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be in an integer-multiple relationship with *data_in_width*. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
depth	2 to 2^{24}	Number of memory elements used in the FIFO (addr_width = ceil[$\log_2$(*depth*)])
ae_level	1 to *depth* – 1	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_level	1 to *depth* – 1	Almost full level (the number of empty memory locations in the FIFO at which the almost_full flag is active.
err_mode	0 to 2 Default: 1	Error mode 0 = underflow/overflow with pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking).
rst_mode	0 or 1 Default: 1	Reset mode 0 = asynchronous reset, 1 = synchronous reset).
byte_order	0 or 1 Default: 0	Order of bytes or subword [*subword* < 8 bits > *subword*] within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position).

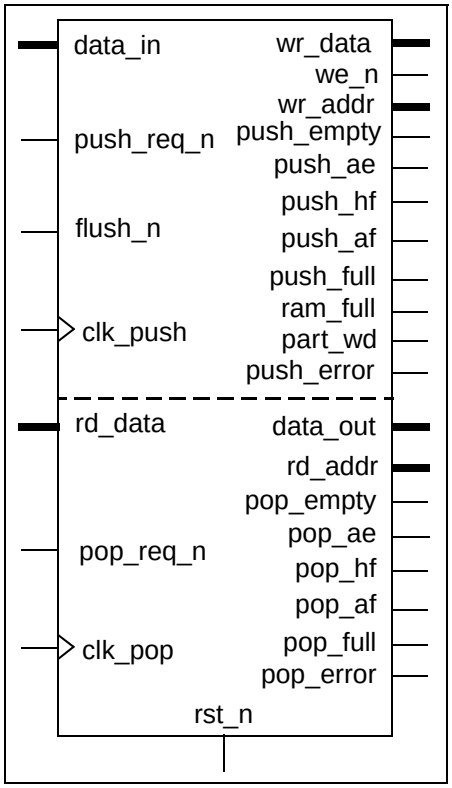
Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Components
cl1	Partial carry look-ahead model	DesignWare-Components
cl2	Full carry look-ahead model	DesignWare-Components

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW_asymfifoctl_s2_sf

Asymmetric Synchronous (Dual-Clock) FIFO Controller with Static Flags



- Parameterized asymmetric input and output bit widths (must be integer-multiple relationship)
- Parameterized word depth
- Fully registered synchronous flag output ports
- Separate status flags for each clock domain
- FIFO empty, half full, and full flags
- Parameterized almost full and almost empty flags
- FIFO push error (overflow) and pop error (underflow) flags
- Single clock cycle push and pop operations
- Parameterized byte order within a word
- Word integrity flag for *data_in_width* < *data_out_width*
- Partial word flush for *data_in_width* < *data_out_width*
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk_push	1 bit	Input	Input clock for push interface
clk_pop	1 bit	Input	Input clock for pop interface
rst_n	1 bit	Input	Reset input, active low
push_req_n	1 bit	Input	FIFO push request, active low
flush_n	1 bit	Input	Flushes the partial word into memory (fills in 0's) (for *data_in_width* < *data_out_width* only)
pop_req_n	1 bit	Input	FIFO pop request, active low
data_in	*data_in_width* bit(s)	Input	FIFO data to push
rd_data	max (*data_in_width*, *data_out_width*) bit(s)	Input	RAM data input to FIFO controller

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
we_n	1 bit	Output	Write enable output for write port of RAM, active low
push_empty	1 bit	Output	FIFO empty [a] output flag synchronous to clk_push, active high
push_ae	1 bit	Output	FIFO almost empty[a] output flag synchronous to clk_push, active high (determined by push_ae_lvl parameter)
push_hf	1 bit	Output	FIFO half full[a] output flag synchronous to clk_push, active high
push_af	1 bit	Output	FIFO almost full[a] output flag synchronous to clk_push, active high (determined by push_af_lvl parameter)
push_full	1 bit	Output	FIFO's RAM full[a] output flag (including the input buffer of FIFO controller for *data_in_width* < *data_out_width*) synchronous to clk_push, active high
ram_full	1 bit	Output	FIFO's RAM (excluding the input buffer of FIFO controller for *data_in_width* < *data_out_width*) full output flag synchronous to clk_push, active high
part_wd	1 bit	Output	Partial word accumulated in the input buffer synchronous to clk_push, active high (for *data_in_width* < *data_out_width* only; otherwise, tied low)
push_error	1 bit	Output	FIFO push error (overrun) output flag synchronous to clk_push, active high
pop_empty	1 bit	Output	FIFO empty [b] output flag synchronous to clk_pop, active high
pop_ae	1 bit	Output	FIFO almost empty[b] output flag synchronous to clk_pop, active high (determined by pop_ae_lvl parameter)
pop_hf	1 bit	Output	FIFO half full[b] output flag synchronous to clk_pop, active high
pop_af	1 bit	Output	FIFO almost full[b] output flag synchronous to clk_pop, active high (determined by pop_af_lvl parameter)

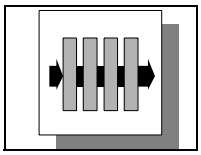


Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
pop_full	1 bit	Output	FIFO's RAM full[b] output flag (excluding the input buffer of FIFO controller for case *data_in_width* < *data_out_width*) synchronous to clk_pop, active high
pop_error	1 bit	Output	FIFO pop error (underrun) output flag synchronous to clk_pop, active high
wr_data	max (*data_in_width*, *data_out_width*) bit(s)	Output	FIFO controller output data to RAM
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM
data_out	*data_out_width* bit(s)	Output	FIFO data to pop

a. As perceived by the push interface.

b. As perceived by the pop interface.

Table 2: Parameter Description

Parameter	Values	Description
data_in_width	1 to 256	Width of the data_in bus. data_in_width must be in an integer-multiple relationship with data_out_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
data_out_width	1 to 256	Width of the data_out bus. data_out_width must be in an integer-multiple relationship with data_in_width. That is, either *data_in_width* = *K* × *data_out_width*, or *data_out_width* = *K* × *data_in_width*, where *K* is an integer.
depth	4 to 2^{24}	Number of words that can be stored in FIFO
push_ae_lvl	1 to *depth* – 1	Almost empty level for the push_ae output port (the number of words in the FIFO at or below which the push_ae flag is active)
push_af_lvl	1 to *depth* – 1	Almost full level for the push_af output port (the number of empty memory locations in the FIFO at which the push_af flag is active)
pop_ae_lvl	1 to *depth* – 1	Almost empty level for the pop_ae output port (the number of words in the FIFO at or below which the pop_ae flag is active)
pop_af_lvl	1 to *depth* – 1	Almost full level for the pop_af output port (the number of empty memory locations in the FIFO at which the pop_af flag is active)
err_mode	0 or 1	Error mode 0 = stays active until reset [latched], 1 = active only as long as error condition exists [unlatched])

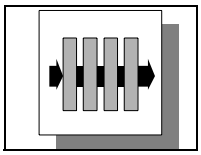
Table 2: Parameter Description (Continued)

Parameter	Values	Description
push_sync	1 to 3	Push flag synchronization mode 1 = single register synchronization from pop pointer, 2 = double register, 3 = triple register)
pop_sync	1 to 3	Pop flag synchronization mode 1 = single register synchronization from push pointer, 2 = double register, 3 = triple register)
rst_mode	0 or 1	Reset mode 0 = asynchronous reset, 1 = synchronous reset)
byte_order	0 or 1 Default: 0	Order of bytes or subword within a word 0 = first byte is in most significant bits position; 1 = first byte is in the least significant bits position).

Table 3: Synthesis Implementations[a]

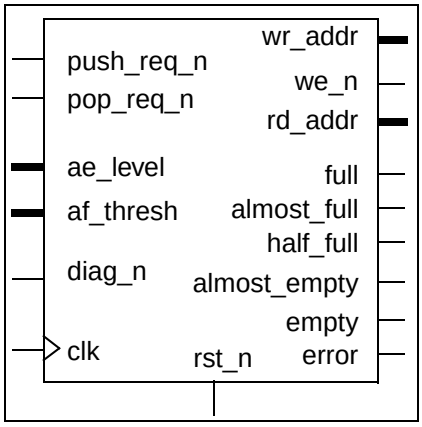
Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_fifoctl_s1_df

Synchronous (Single Clock) FIFO Controller with Dynamic Flags



- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Dynamically programmable almost full and almost empty flags
- Parameterized word depth
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0, synchronous if rst_mode = 1
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control for err_mode = 0, NC for other err_mode values, active low
ae_level	ceil($\log_2$[*depth*]) bit(s)	Input	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)

Table 1: Pin Description

Pin Name	Width	Direction	Function
af_thresh	ceil($\log_2$[*depth*]) bit(s)	Input	Almost full threshold (the number of words stored in the FIFO at or above which the almost_full flag is active)
we_n	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM

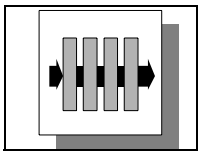
Table 2: Parameter Description

Parameter	Values	Description
depth	2 to 2^{24}	Number of memory elements used in FIFO [used to size the address ports]
err_mode	0 to 2 Default: 0	Error mode 0 = underflow/overflow and pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset

Table 3: Synthesis Implementations[a]

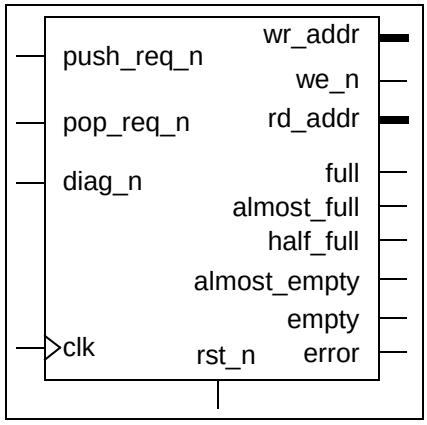
Implementation Name	Function	License Required
rpl	Ripple Carry synthesis model	DesignWare-Foundation
cl1	Partial Carry lookahead model	DesignWare-Foundation
cl2	Full Carry lookahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_fifoctl_s1_sf

Synchronous (SingleClock) FIFO Controller with Static Flags



- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0, synchronous if rst_mode = 1
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control for err_mode = 0, NC for other err_mode values), active low
we_n	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, asserted when FIFO level ≤*ae_level*, active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, asserted when FIFO level ≥ (*depth* – *af_leve*l), active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM

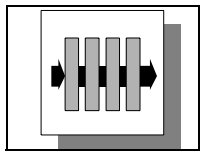
Table 2: Parameter Description

Parameter	Values	Function
depth	2 to 2^{24} Default: 4	Number of memory elements used in FIFO (used to size the address ports)
ae_level	1 to *depth* – 1 Default: 1	Almost empty level (the number of words in the FIFO at or below which the almost_empty flag is active)
af_level	1 to *depth* – 1 Default: 1	Almost full level (the number of empty memory locations in the FIFO at which the almost_full flag is active.)
err_mode	0 to 2 Default: 0	Error mode 0 = underflow/overflow and pointer latched checking, 1 = underflow/overflow latched checking, 2 = underflow/overflow unlatched checking
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl1	Partial carry look-ahead model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_fifoctl_s2_sf

Synchronous (Dual Clock) FIFO Controller with Static Flags

- Fully registered synchronous flag output ports
- Single clock cycle push and pop operations
- Separate status flags for each clock system
- FIFO empty, half full, and full flags
- FIFO push error (overflow) and pop error (underflow) flags
- Parameterized word depth
- Parameterized almost full and almost empty flag thresholds
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs

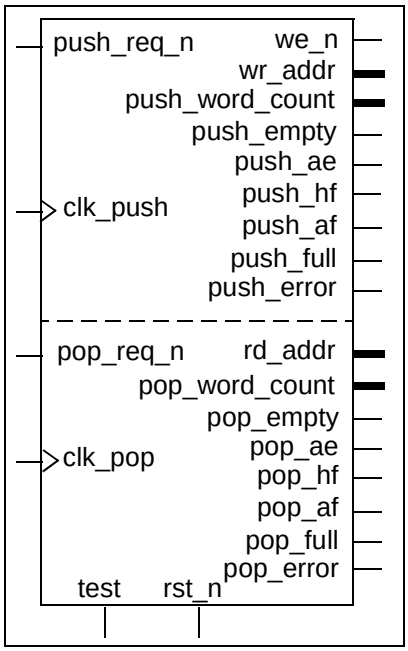


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk_push	1 bit	Input	Input clock for push interface
clk_pop	1 bit	Input	Input clock for pop interface
rst_n	1 bit	Input	Reset input, active low
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
we_n	1 bit	Output	Write enable output for write port of RAM, active low
push_empty	1 bit	Output	FIFO empty[a] output flag synchronous to clk_push, active high
push_ae	1 bit	Output	FIFO almost empty[a] output flag synchronous to clk_push, active high (determined by push_ae_lvl parameter)
push_hf	1 bit	Output	FIFO half full[a] output flag synchronous to clk_push, active high
push_af	1 bit	Output	FIFO almost full[a] output flag synchronous to clk_push, active high (determined by push_af_lvl parameter)

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
push_full	1 bit	Output	FIFO full[a] output flag synchronous to clk_push, active high
push_error	1 bit	Output	FIFO push error (overrun) output flag synchronous to clk_push, active high
pop_empty	1 bit	Output	FIFO empty[b] output flag synchronous to clk_pop, active high
pop_ae	1 bit	Output	FIFO almost empty[b] output flag synchronous to clk_pop, active high (determined by pop_ae_lvl parameter)
pop_hf	1 bit	Output	FIFO half full[b] output flag synchronous to clk_pop, active high
pop_af	1 bit	Output	FIFO almost full[b] output flag synchronous to clk_pop, active high (determined by pop_af_lvl parameter)
pop_full	1 bit	Output	FIFO full[b] output flag synchronous to clk_pop, active high
pop_error	1 bit	Output	FIFO pop error (underrun) output flag synchronous to clk_pop, active high
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM
push_word_count	ceil($\log_2$[*depth*+1]) bit(s)	Output	Words in FIFO (as perceived by the push/pop interface)
pop_word_count	ceil($\log_2$[*depth*+1]) bit(s)	Output	Words in FIFO (as perceived by the push/pop interface)
test	1 bit	Input	Active high, test input control for inserting scan test lock-up latches

a. As perceived by the push interface.

b. As perceived by the pop interface.

Table 2: Parameter Description

Parameter	Values	Description
depth	4 to 2^{24} Default: 8	Number of words that can be stored in FIFO
push_ae_lvl	1 to *depth* – 1 Default: 2	Almost empty level for the push_ae output port (the number of words in the FIFO at or below which the push_ae flag is active)

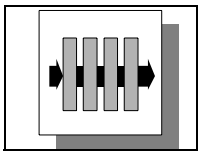


Table 2: Parameter Description (Continued)

Parameter	Values	Description
push_af_lvl	1 to *depth* – 1 Default: 2	Almost full level for the push_af output port (the number of empty memory locations in the FIFO at which the push_af flag is active)
pop_ae_lvl	1 to *depth* – 1 Default: 2	Almost empty level for the pop_ae output port (the number of words in the FIFO at or below which the pop_ae flag is active)
pop_af_lvl	1 to *depth* – 1 Default: 2	Almost full level for the pop_af output port (the number of empty memory locations in the FIFO at which the pop_af flag is active)
err_mode	0 or 1 Default: 0	Error mode 0 = stays active until reset [latched], 1 = active only as long as error condition exists [unlatched]
push_sync	1 to 3 Default: 2	Push flag synchronization mode 1 = single register synchronization from pop pointer, 2 = double register, 3 = triple register)
pop_sync	1 to 3 Default: 2	Pop flag synchronization mode 1 = single register synchronization from push pointer, 2 = double register, 3 = triple register
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset)
tst_mode	0 or 1 Default: 0	Test Mode 0 = test input not connected 1 = lock-up latches inserted for scan test

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple Carry synthesis model	DesignWare-Foundation
cl2	Full Carry lookahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

Memory – Synchronous RAMs

This section documents the various memory synchronous RAMs DesignWare Building Block IP.

DW_ram_r_w_s_dff

Synchronous Write-Port, Asynchronous Read-Port RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Parameterized reset mode (synchronous or asynchronous)
- Inferable from Behavioral Compiler
- High testability using DFT Compiler

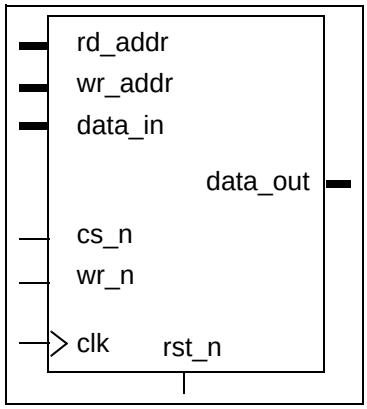


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines the reset methodology: 0 = rst_n asynchronously initializes the RAM, 1 = rst_n synchronously initializes the RAM

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_r_w_s_lat

Synchronous Write Port, Asynchronous Read Port RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Inferable from Behavioral Compiler

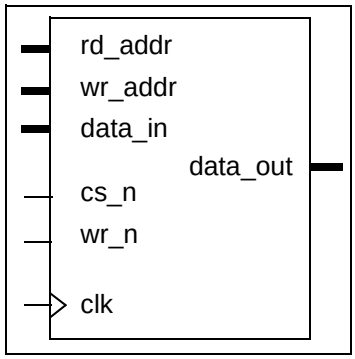


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_2r_w_s_dff

Synchronous Write Port, Asynchronous Dual Read Port RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Parameterized reset mode (synchronous or asynchronous)
- Inferable from Behavioral Compiler
- High testability using DFT Compiler

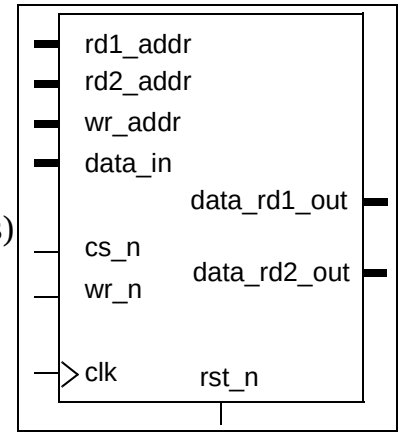


Table 1: Pin Description

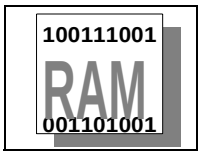
Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd1_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read1 address bus
rd2_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read2 address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_rd1_out	*data_width* bit(s)	Output	Output data bus for read1
data_rd2_out	*data_width* bit(s)	Output	Output data bus for read2

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines the reset methodology: 0 = rst_n asynchronously initializes the RAM, 1 = rst_n synchronously initializes the RAM

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW_ram_2r_w_s_lat

Synchronous Write Port, Asynchronous Dual Read Port RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Inferable from Behavioral Compiler

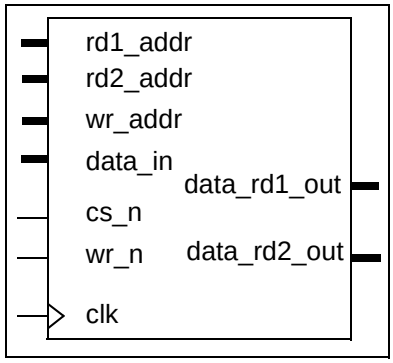


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd1_addr	ceil($\log_2$[*depth*]) bit	Input	Read1 address bus
rd2_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read2 address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_rd1_out	*data_width* bit(s)	Output	Output data bus for read1
data_rd2_out	*data_width* bit(s)	Output	Output data bus for read2

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_rw_s_dff

Synchronous Single Port Read/Write RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Parameterized reset mode (asynchronous or synchronous)
- Inferable by Behavioral Compiler
- High testability using DFT Compiler

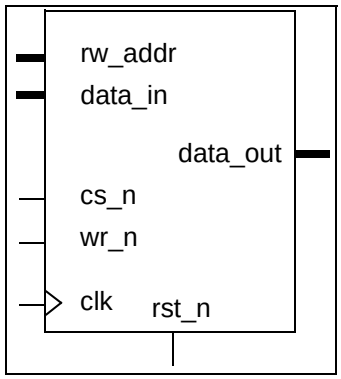


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rw_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines the reset methodology: 0 = rst_n asynchronously initializes the RAM, 1 = rst_n synchronously initializes the RAM

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Components

DW_ram_rw_s_lat

Synchronous Single Port Read/Write RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Synchronous static memory
- Inferable from Behavioral Compiler

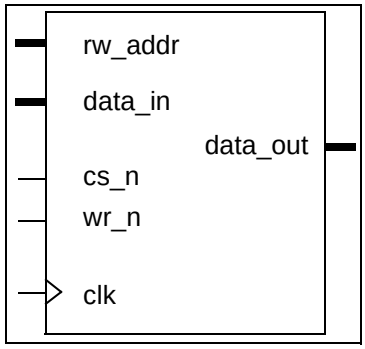


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rw_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Components

Memory – Asynchronous RAMs

This section documents the various memory asynchronous RAMs found in the library of DesignWare Building Block IP.

DW_ram_r_w_a_dff

Asynchronous Dual Port RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation
- High testability using DFT Compiler

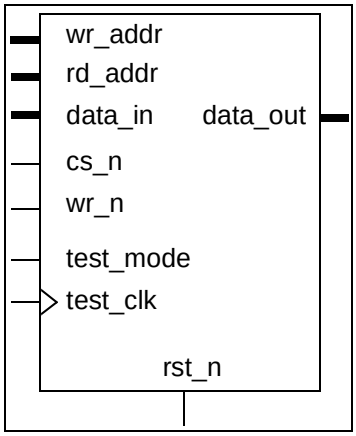


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
test_mode	1 bit	Input	Enables test_clk
test_clk	1 bit	Input	Test clock to capture data during test_mode
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0 = rst_n initializes the RAM, 1 = rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_r_w_a_lat

Asynchronous Dual Port RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation

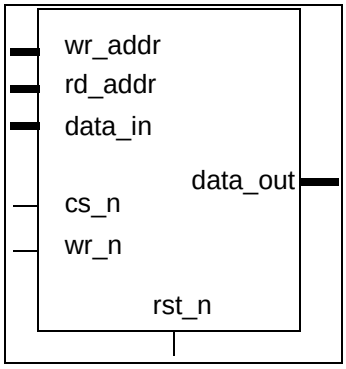


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0= rst_n initializes the RAM, 1= rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_2r_w_a_dff

Write Port, Dual Read Port RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation
- High testability using DFT Compiler

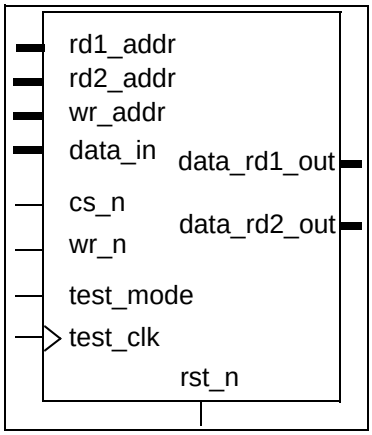


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
test_mode	1 bit	Input	Enables test_clk
test_clk	1 bit	Input	Test clock to capture data during test_mode
rd1_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read1 address bus
rd2_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Read2 address bus
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Write address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_rd1_out	*data_width* bit(s)	Output	Output data bus for read1
data_rd2_out	*data_width* bit(s)	Output	Output data bus for read2

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0 = rst_n initializes the RAM, 1 = rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_2r_w_a_lat

Write Port, Dual Read Port RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation

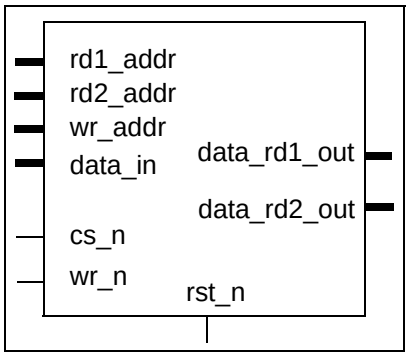


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rd1_addr	ceil($\log_2$[*depth*]) bit	Input	Read1 address bus
rd2_addr	ceil($\log_2$[*depth*]) bit	Input	Read2 address bus
wr_addr	ceil($\log_2$[*depth*]) bit	Input	Write address bus
data_in	*data_width* bit	Input	Input data bus
data_rd1_out	*data_width* bit	Output	Output data bus for read1
data_rd2_out	*data_width* bit	Output	Output data bus for read2

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0 = rst_n initializes the RAM, 1 = rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_rw_a_dff

Asynchronous Single Port RAM (Flip-Flop-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation
- High testability using DFT Compiler

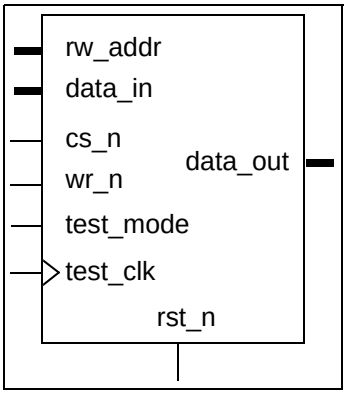


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
test_mode	1 bit	Input	Enables test_clk
test_clk	1 bit	Input	Test clock to capture data during test_mode
rw_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0 = rst_n initializes the RAM, 1 = rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_ram_rw_a_lat

Asynchronous Single-Port RAM (Latch-Based)

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation

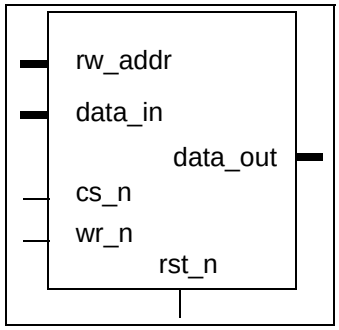


Table 1: Pin Description

Pin Name	Width	Direction	Function
rst_n	1 bit	Input	Reset, active low
cs_n	1 bit	Input	Chip select, active low
wr_n	1 bit	Input	Write enable, active low
rw_addr	ceil($\log_2$[*depth*]) bit(s)	Input	Address bus
data_in	*data_width* bit(s)	Input	Input data bus
data_out	*data_width* bit(s)	Output	Output data bus

Table 2: Parameter Description

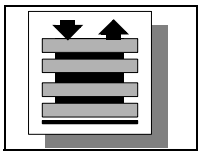
Parameter	Values	Description
data_width	1 to 256	Width of data_in and data_out buses
depth	2 to 256	Number of words in the memory array (address width)
rst_mode	0 or 1	Determines if the rst_n input is used. 0 = rst_n initializes the RAM, 1 = rst_n is not connected

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

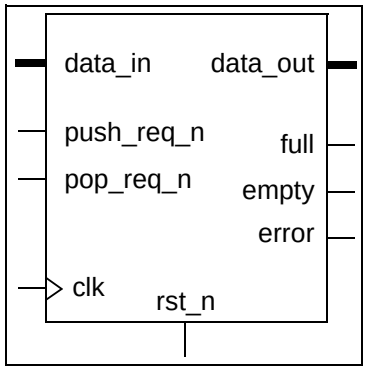
Memory – Stacks

This section documents the various memory stacks found in the library of DesignWare Building Block IP.



DW_stack

Synchronous (Single-Clock) Stack



- Parameterized word width and depth
- Stack empty and full status flags
- Stack error flag indicating underflow and overflow
- Fully registered synchronous flag output ports
- All operations execute in a single clock cycle
- D flip-flop based memory array for high testability
- Parameterized reset mode (synchronous or asynchronous)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0 or 2, synchronous if rst_mode = 1 or 3
push_req_n	1 bit	Input	Stack push request, active low
pop_req_n	1 bit	Input	Stack pop request, active low
data_in	*data_width* bit(s)	Input	Stack push data
empty	1 bit	Output	Stack empty flag, active high
full	1 bit	Output	Stack full flag, active high
error	1 bit	Output	Stack error output, active high
data_out	*data_width* bit(s)	Output	Stack pop data

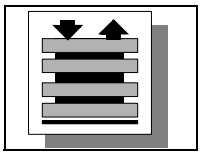
Table 2: Parameter Description

Parameter	Values	Description
width	1 to 256 Default: None	Width of data_in and data_out buses
depth	2 to 256 Default: None	Depth (in words) of memory array
err_mode	0 or 1 Default: 0	Error mode 0 = underflow/overflow error, hold until reset, 1 = underflow/overflow error, hold until next clock.
rst_mode	0 to 3 Default: 0	Reset mode 0 = asynchronous reset including memory, 1 = synchronous reset including memory, 2 = asynchronous reset excluding memory, 3 = synchronous reset excluding memory.

Table 3: Synthesis Implementations[a]

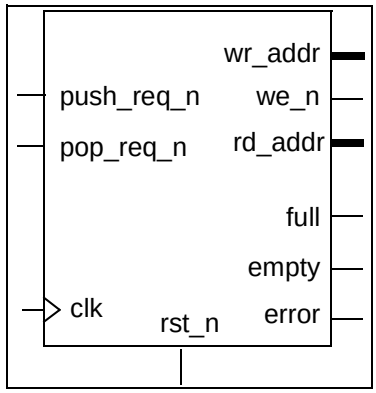
Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_stackctl

Synchronous (Single-Clock) Stack Controller



- Parameterized word width and depth
- Stack empty and full status flags
- Stack error flag indicating underflow and overflow
- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces with common hard macro or compiled ASIC dual-port synchronous RAMs

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if rst_mode = 0, synchronous if rst_mode = 1
push_req_n	1 bit	Input	Stack push request, active low
pop_req_n	1 bit	Input	Stack pop request, active low
we_n	1 bit	Output	Write enable for RAM write port, active low
empty	1 bit	Output	Stack empty flag, active high
full	1 bit	Output	Stack full flag, active high
error	1 bit	Output	Stack error output, active high
wr_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to write port of RAM
rd_addr	ceil($\log_2$[*depth*]) bit(s)	Output	Address output to read port of RAM

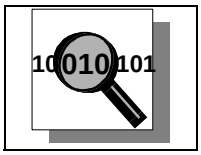
Table 2: Parameter Description

Parameter	Values	Function
depth	2 to 2^{24} Default: None	Number of memory elements in the stack [used to size the address ports]
err_mode	0 or 1 Default: 0	Error mode 0 = underflow/overflow error, hold until reset, 1 = underflow/overflow error, hold until next clock.
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset, 1 = synchronous reset.

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cl2	Full carry look-ahead model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

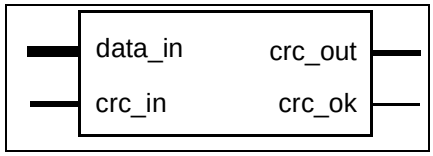


Data Integrity

This section documents the various data integrity IP found in the DesignWare Building Block IP.

DW_crc_p

Universal Parallel (Combinational) CRC Generator/Checker



- Parameterized arbitrary polynomial (up to 64-bit)
- Parameterized data width (up to 512 bits)
- Parameterized initial CRC value (all ones or all zeroes)
- Parameterized inversion of generated CRC
- Parameterized bit and byte ordering

Table 1: Pin Description

Pin Name	Width	Direction	Function
data_in	*data_width* bit(s)	Input	Input data used for both generating and checking for valid CRC
crc_in	*poly_size* bit(s)	Input	Input CRC value used to check a record (not used when generating CRC from data_in)
crc_ok	1 bit	Output	Indicates a correct residual CRC value, active high
crc_out	*poly_size* bit(s)	Output	Provides the CRC check bits to be appended to the input data to form a valid record (data_in and crc_in)

Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to 512 Default: 16	Width of data_in (i.e. the amount of data that CRC will be calculated upon)
poly_size	2 to 64 Default: 16	Size of the CRC polynomial and thus the width of crc_in and crc_out
crc_cfg	0 to 7 Default: 7	CRC initialization and insertion configuration
bit_order	0 to 3 Default: 3	Bit and byte order configuration
poly_coef0	1 to 65535[a] Default: 4129[b]	Polynomial coefficients 0 through 15
poly_coef1	0 to 65535 Default: 0	Polynomial coefficients 16 through 31

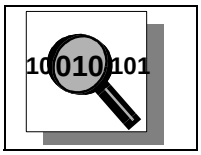


Table 2: Parameter Description (Continued)

Parameter	Values	Description
poly_coef2	0 to 65535 Default: 0	Polynomial coefficients 32 through 47
poly_coef3	0 to 65535 Default: 0	Polynomial coefficients 48 through 63

a. poly_coef0 must be an odd number (since all primitive polynomials include the coefficient 1, which is equivalent to X^0).

b. CCITT-CRC16 polynomial is X16 + X12 + X5 + 1, thus poly_coef0 = 212 + 25 + 1 = 4129.

Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str	Synthesis model	DesignWare-Foundation

DW_crc_s

Universal Synchronous (Clocked) CRC Generator/Checker

- Parameterized arbitrary polynomial (up to 64-bit)
- Parameterized data width (up to polynomial size)
- Parameterized register initialization (all ones or all zeroes)
- Parameterized inverted insertion of generated CRC
- Parameterized bit and byte ordering
- Loadable CRC value for use in context switching of interspersed blocks

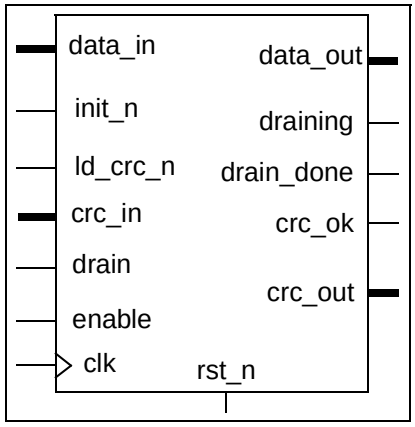


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock input
rst_n	1 bit	Input	Asynchronous reset input, active low
init_n	1 bit	Input	Synchronous initialization control input, active low
enable	1 bit	Input	Enable control input for all operations (other than reset and initialization), active high
drain	1 bit	Input	Drains control input, active high
ld_crc_n	1 bit	Input	Synchronous CRC register load control input, active low
data_in	*data_width* bit(s)	Input	Input data
crc_in	*poly_size* bit(s)	Input	Input CRC value (to be loaded into the CRC register as commanded by the ld_crc_n control input)
draining	1 bit	Output	Indicates that the CRC register is draining (inserting the CRC into the data stream)
drain_done	1 bit	Output	Indicates that the CRC register has finished draining
crc_ok	1 bit	Output	Indicates a correct residual CRC value, active high
data_out	*data_width* bit(s)	Output	Output data
crc_out	*poly_size* bit(s)	Output	Provides constant monitoring of the CRC register

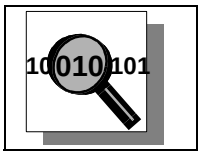


Table 2: Parameter Description

Parameter	Values	Description
data_width	1 to *poly_size*[a] Default: 16	Width of data_in and data_out (also the number of bits per clock)
poly_size	2 to 64 Default: 16	Size of the CRC polynomial
crc_cfg	0 to 7 Default: 7	CRC initialization and insertion configuration
bit_order	0 to 3 Default: 3	Bit and byte order configuration
poly_coef0	1 to 65535[b] Default: 4129[c]	Polynomial coefficients 0 through 15
poly_coef1	0 to 65535 Default: 0	Polynomial coefficients 16 through 31
poly_coef2	0 to 65535 Default: 0	Polynomial coefficients 32 through 47
poly_coef3	0 to 65535 Default: 0	Polynomial coefficients 48 through 63

a. The *data_width* value must be chosen such that *poly_size* is a multiple of *data_width*.

b. The *poly_coef0* value must be an odd number (since all primitive polynomials include the coefficient, which is equivalent to X^0).

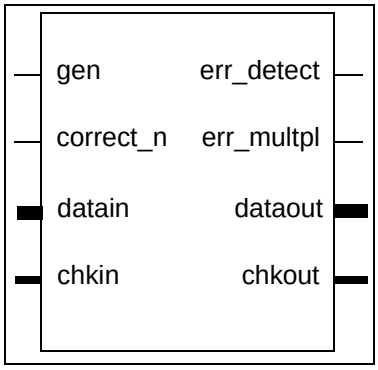
c. CCITT-CRC16 polynomial is $X^{16} + X^{12} + X^5 + 1$, thus poly_coef0 = $2^{12} + 2^5 + 1 = 4129$.

Table 3: Synthesis Implementations

Implementation Name	Implementation	License Required
str	Synthesis model	DesignWare-Foundation

DW_ecc

Error Checking and Correction



- Parameterized word width
- Generates check bits for new data written, and corrects corrupt data for read and read-modify-write cycles
- Supports scrubbing
- Flags to indicate if an error was detected, and if the error is not correctable
- Flow-through architecture for speed and flexibility
- Error syndrome output for error logging

Table 1: Pin Description

Pin Name	Width	Direction	Function
gen	1 bit	Input	Suppresses correction in write mode (gen = 1) and generates check bits. Enables correction when in read mode (gen = 0) and correct_n is asserted (low).
correct_n	1 bit	Input	Enables correction of correctable words, active low
datain	*width* bits	Input	Input data word to check (check mode), or data from which check bits are generated (generate mode)
chkin	*chkbits* bits	Input	Check bits input for error analysis on read
err_detect	1 bit	Output	Indicates that an error has been detected, active high. Location of error is specified by the error syndrome.
err_multpl	1 bit	Output	Indicates that the error detected is a multiple-bit error and, therefore, uncorrectable
dataout	*width* bits	Output	Output data. May be corrected if an error is detected and correct_n is asserted.
chkout	*chkbits* bits	Output	When gen = 1, chkout contains the check bits generated from datain. When gen = 0 and synd_sel = 0, chkout is the corrected or uncorrected data from chkin. When gen = 0 and synd_sel= 1, chkout is the error syndrome value

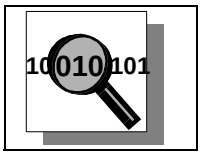


Table 2: Parameter Description

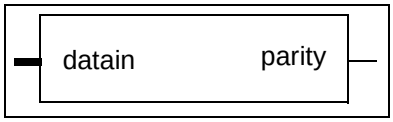
Parameter	Values	Description
width	8 to 502	Width of input and output data buses
chkbits	5 to 10	Width of check bits input and output buses, calculated from *width*
synd_sel	0 or 1	Selects function of chkout when gen = 0. If synd_sel = 0 and gen = 0, then chkout is the corrected or uncorrected data from chkin. If synd_sel = 1 and gen = 0, then chkout is the error syndrome value

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW04_par_gen

Parity Generator and Checker



- Generates parity for given input data
- Supports even and odd parity, selectable via a parameter
- Supports variable word widths
- Inferable using a function call

Table 1: Pin Description

Pin Name	Width	Direction	Function
datain	*width* bit(s)	Input	Input data word to check or generate parity
parity	1 bit	Output	Generated parity

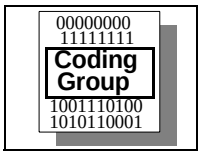
Table 2: Parameter Description

Parameter	Values[a]	Description
width	1 to 256	Defines the width of the input bus
par_type	0 or 1	Defines the type of parity

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



Data Integrity – Coding Group Overview

The Coding Group consists of a set of IP that encode and/or decode data for use in data communications and data storage applications. Currently the 8B/10B coding scheme (used in standard data communication and networking protocols such as Gigabit Ethernet and Fiber Channel) is embodied in the Coding Group IP.

DW_8b10b_dec

8b10b Decoder

- Configurable data width
- Configurable simplified Special Character indicator flags (for protocols requiring only the K28.5 special character)
- Synchronous initialization of Running Disparity with design specified value
- All outputs registered

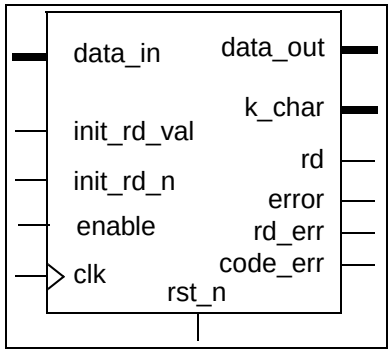


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock input
rst_n	1 bit	Input	Asynchronous reset input, active low
init_rd_n	1 bit	Input	Synchronous initialization control input, active low
init_rd_val	1 bit	Input	Value of initial Running Disparity
data_in	*bytes* × 10 bit(s)	Input	Input 8b/10b data for decoding
error	1 bit	Output	Active high, error flag indicating the presence of any type of error (running disparity or coding) in the information currently decoded on data_out
rd	1 bit	Output	Current Running Disparity (after decoding data presented at data_in to data_out)
k_char	*bytes* bit(s)	Output	Special Character indicators (one indicator per decoded byte)
data_out	*bytes* × 8 bit(s)	Output	Decoded output data
rd_err	1 bit	Output	Active high, error flag indicating the presence of one or more Running Disparity errors in the information currently decoded on data_out
code_err	1 bit	Output	Active high, error flag indicating the presence of a coding error in at least one byte of information currently decoded on data_out
enable	1 bit	Input	Enables register clocking

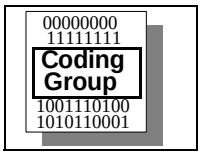


Table 2: Parameter Description

Parameter	Values	Description
bytes	1 to 16 Default: 2	Number of bytes to encode
k28_5_only	0 or 1 Default: 0	Special Character subset control parameter 0 - for all special characters decoded, 1 - for only K28.5 decoded [when k_char = HIGH implies K28.5, all other special characters indicate an error]
en_mode	0 or 1 Default:0	Enable control 0 - the enable input port is not connected (backward compatible with older components) 1 - when enable=0 the decoder is stalled

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
rtl	Synthesis model	DesignWare-Foundation

DW_8b10b_enc

8b10b Encoder

- Configurable data width
- Configurable simplified Special Character control (for protocols requiring only the K28.5 special character)
- Synchronous initialization of Running Disparity with design specified value
- All outputs registered

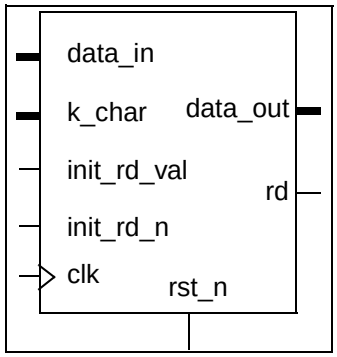


Table 1: Pin Description

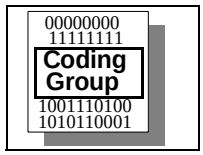
Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Asynchronous reset, active low
init_rd_n	1 bit	Input	Synchronous initialization, active low
init_rd_val	1 bit	Input	Value of initial Running Disparity
k_char	*bytes* bit(s)	Input	Special Character controls (one control per byte to encode)
data_in	*bytes* × 8 bit(s)	Input	Input data for encoding
rd	1 bit	Output	Current Running Disparity (before encoding data presented at data_in)
data_out	*bytes* × 10 bit(s)	Output	8b10b Encoded data

Table 2: Parameter Description

Parameter	Value	Description
bytes	1 to 16 Default: 2	Number of bytes to encode
k28_5_only	0 or 1 Default: 0	Special Character subset control parameter 0 for all special characters available, 1 for only K28.5 available [when k_char = HIGH, regardless of the value on data_in]

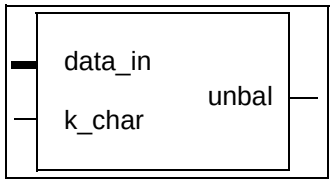
Table 3: Synthesis Implementations

Implementation	Function	License Required
rtl	Synthesis model	DesignWare-Foundation



DW_8b10b_unbal

8b10b Coding Balance Predictor



- Independent of Running Disparity
- Higher speed than a full encoder
- Predicts balance for both data and special characters

Table 1: Pin Description

Pin Name	Width	Direction	Function
k_char	1 bit	Input	Special character control input (LOW for data characters, HIGH for special characters)
data_in	8 bits	Input	Input for 8-bit data character to be encoded
unbal	1 bit	Output	Unbalanced code character indicator (LOW for balanced, HIGH for unbalanced)

Table 2: Parameter Description

Parameter	Values	Description
k28_5_mode	0 or 1 Default: 0	Special Character subset control parameter 0 for all special characters available, 1 for only K28.5 available [when k_char = HIGH, regardless of the value on data_in]

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
rtl	Synthesis model	DesignWare-Foundation

Test – JTAG Overview

The JTAG IP consist of a set of boundary scan IP. The boundary scan IP include a parameterized Test Access Port (TAP) controller (DW_tap) plus a set of boundary scan cells that you can use to implement a custom IEEE 1149.1 boundary scan test solution for your ASIC.

DW_tap
TAP Controller

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous registers with respect to `tck`
- Supports the standard instructions EXTEST, SAMPLE/PRELOAD, and BYPASS
- Supports the optional instructions IDCODE, INTEST, RUNBIST, CLAMP, and HIGHZ
- Optional use of device identification register and IDCODE instruction
- Parameterized instruction register width

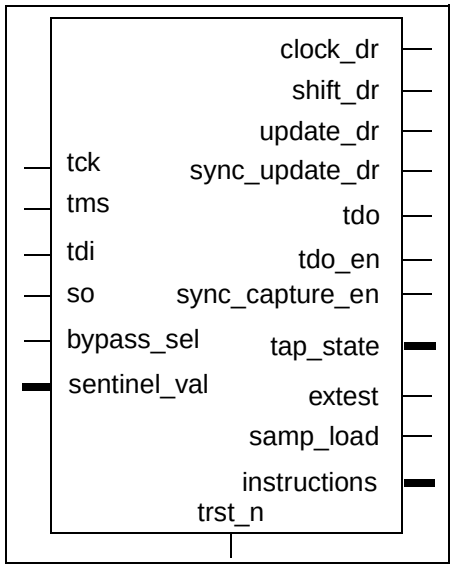


Table 1: Pin Description

Pin Name	Width	Direction	Function
tck	1 bit	Input	Test clock
trst_n	1 bit	Input	Test reset, active low
tms	1 bit	Input	Test mode select
tdi	1 bit	Input	Test data in
so	1 bit	Input	Serial data from boundary scan register and data registers
bypass_sel	1 bit	Input	Selects the bypass register, active high
sentinel_val	*width* –1 bit(s)	Input	User-defined status bits
clock_dr	1 bit	Output	Clocks in data in asynchronous mode
shift_dr	1 bit	Output	Enables shifting of data in both synchronous and asynchronous mode
update_dr	1 bit	Output	Enables updating data in asynchronous mode
tdo	1 bit	Output	Test data out
tdo_en	1 bit	Output	Enable for tdo output buffer
tap_state	16 bits	Output	Current state of the TAP finite state machine
extest	1 bit	Output	EXTEST decoded instruction
samp_load	1 bit	Output	SAMPLE/PRELOAD decoded instruction
instructions	*width* bit(s)	Output	Instruction register output

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
sync_capture_en	1 bit	Output	Enable for synchronous capture
sync_update_dr	1 bit	Output	Enables updating new data in synchronous_mode

Table 2: Parameter Description

Parameter	Values	Description
width	2 to 32 Default: None	Width of instruction register
id	0 or 1 Default: 0	Determines whether the device identification register is present 0 = not present, 1 = present
version	0 to 15 Default: 0	4-bit version number
part	0 to 65535 Default: 0	16-bit part number
man_num	0 to 2047, man_num ≠ 127 Default: 0	11-bit JEDEC manufacturer identity code
sync_mode	0 or 1 Default: 0	Determines whether the bypass, device identification, and instruction registers are synchronous with respect to tck 0 = asynchronous, 1 = synchronous

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_tap_uc

TAP Controller with USERCODE support

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous registers with respect to `tck`
- Provides interface to supports the standard IEEE 1149.1 and optional instructions
- Optional use of device identification register and IDCODE instruction and support of USERCODE instruction
- User defined opcode for IDCODE
- Parameterized instruction register width
- External interface to program device identification register

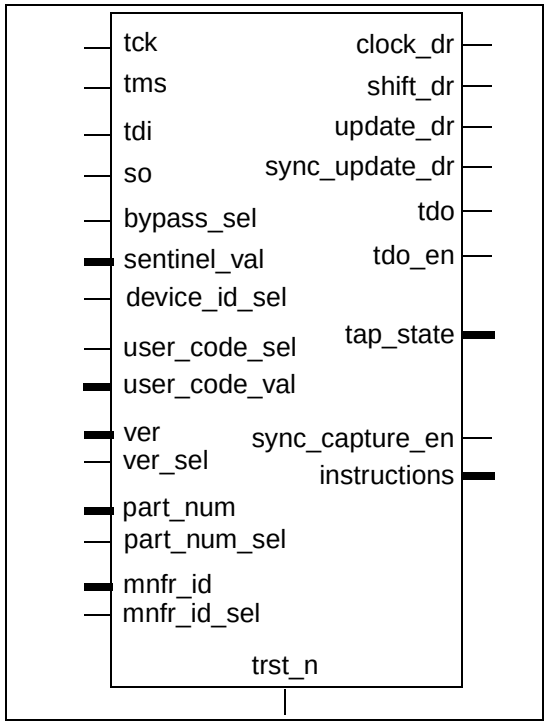


Table 1: Pin Description

Pin Name	Width	Direction	Function
tck	1 bit	Input	Test clock
trst_n	1 bit	Input	Test reset, active low
tms	1 bit	Input	Test mode select
tdi	1 bit	Input	Test data in
so	1 bit	Input	Serial data from boundary scan register and data registers
bypass_sel	1 bit	Input	Selects the bypass register, active high
sentinel_val	*width* –1 bit(s)	Input	User-defined status bits
device_id_sel	1 bit	Input	Selects the device identification register, active high
user_code_sel	1 bit	Input	Selects the user_code_val bus for input in to the device identification register, active high
user_code_val	32 bits	Input	32-bit user defined code.
ver	4 bits	Input	4 bit version number

Table 1: Pin Description (Continued)

Pin Name	Width	Direction	Function
ver_sel	1 bit	Input	Selects version from the parameter or the ver input port 0 = version (parameter) 1 = ver (input port)
part_num	16 bits	Input	16 bit part number
part_num_sel	1 bit	Input	Selects part from the parameter or the part_num from the input port 0 = part (parameter) 1 = part_num (input port)
mnfr_id	11 bits	Input	11 bit JEDEC manufacturer's identity code (mnfr_id ≠ 127)
mnfr_id_sel	1 bit	Input	Selects man_num from the parameter or mnfr_id from the input port 0 = man_num (parameter) 1 = mnfr_id (input port)
clock_dr	1 bit	Output	Clocks in data in asynchronous mode
shift_dr	1 bit	Output	Enables shifting of data in both synchronous and asynchronous mode
update_dr	1 bit	Output	Enables updating data in asynchronous mode
tdo	1 bit	Output	Test data out
tdo_en	1 bit	Output	Enable for tdo output buffer
tap_state	16 bits	Output	Current state of the TAP finite state machine
instructions	*width* bit(s)	Output	Instruction register output
sync_capture_en	1 bit	Output	Enable for synchronous capture
sync_update_dr	1 bit	Output	Enables updating new data in synchronous_mode

Table 2: Parameter Description

Parameter	Values	Description
width	2 to 32 Default: None	Width of instruction register
id	0 or 1 Default: 0	Determines whether the device identification register is present 0 = not present, 1 = present
idcode_opcode	1 to $2^{width-1}$ Default: 1	Opcode for IDCODE.
version	0 to 15 Default: 0	4-bit version number

Table 2: Parameter Description (Continued)

Parameter	Values	Description
part	0 to 65535 Default: 0	16-bit part number
man_num	0 to 2047, man_num ≠ 127 Default: 0	11-bit JEDEC manufacturer identity code
sync_mode	0 or 1 Default: 0	Determines whether the bypass, device identification, and instruction registers are synchronous with respect to tck 0 = asynchronous, 1 = synchronous

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_bc_1

Boundary Scan Cell Type BC_1

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS
- Supports the optional instructions INTEST, RUNBIST, CLAMP, and HIGHZ

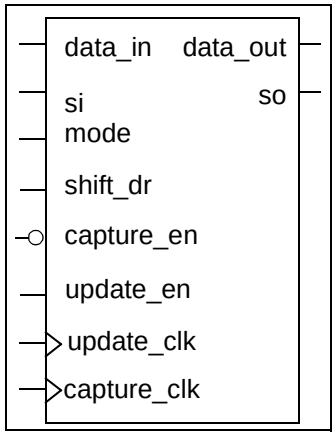


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
update_clk	1 bit	Input	Clocks data into the update stage
capture_en	1 bit	Input	Enable for data clocked into the capture stage, active low
update_en	1 bit	Input	Enable for data clocked into the update stage, active high
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
mode	1 bit	Input	Determines whether data_out is controlled by the boundary scan cell or by the data_in signal
si	1 bit	Input	Serial path from the previous boundary scan cell
data_in	1 bit	Input	Input data
data_out	1 bit	Output	Output data
so	1 bit	Output	Serial path to the next boundary scan cell

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1

DW_bc_2

Boundary Scan Cell Type BC_2

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS
- Supports the optional instructions INTEST, RUNBIST, CLAMP, and HIGHZ

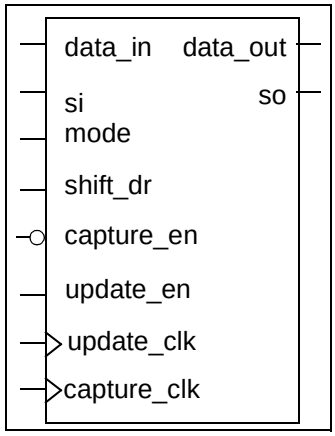


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
update_clk	1 bit	Input	Clocks data into the update stage
capture_en	1 bit	Input	Enable for data clocked into the capture stage, active low
update_en	1 bit	Input	Enable for data clocked into the update stage, active high
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
mode	1 bit	Input	Determines whether data_out is controlled by the boundary scan cell or by the data_in signal
si	1 bit	Input	Serial path from the previous boundary scan cell
data_in	1 bit	Input	Input data
data_out	1 bit	Output	Output data
so	1 bit	Output	Serial path to the next boundary scan cell

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1

DW_bc_3

Boundary Scan Cell Type BC_3

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS
- Supports the optional instructions INTEST, RUNBIST, CLAMP, and HIGHZ

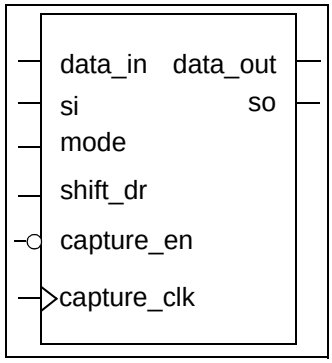


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
capture_en	1 bit	Input	Enable for data clocked into capture stage, active low
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
mode	1 bit	Input	Determines whether data_out is controlled by the boundary scan cell or by the data_in signal
si	1 bit	Input	Serial path from the previous boundary scan cell
data_in	1 bit	Input	Input data from system input pin
data_out	1 bit	Output	Output data to IC logic
so	1 bit	Output	Serial path to the next boundary scan cell

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1

DW_bc_4

Boundary Scan Cell Type BC_4

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS

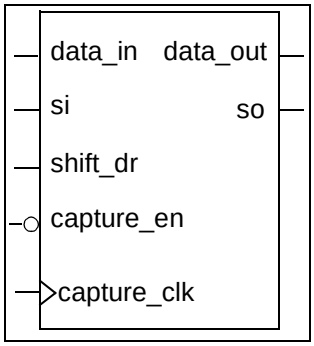


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
capture_en	1 bit	Input	Enable for data clocked into the capture stage, active low
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
si	1 bit	Input	Serial path from the previous boundary scan cell
data_in	1 bit	Input	Input data from system input pin
so	1 bit	Output	Serial path to the next boundary scan cell
data_out	1 bit	Output	Output data

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1

DW_bc_5

Boundary Scan Cell Type BC_5

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS
- Supports the optional instructions INTEST, RUNBIST, CLAMP, and HIGHZ

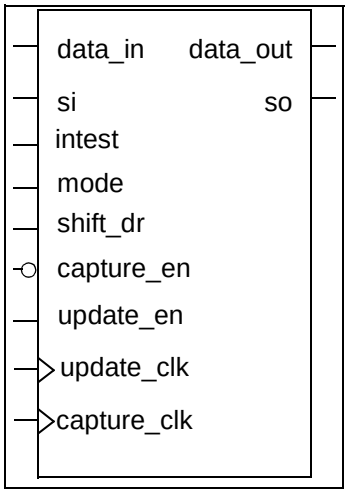


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
update_clk	1 bit	Input	Clocks data into the update stage
capture_en	1 bit	Input	Enable for data clocked into the capture stage, active low
update_en	1 bit	Input	Enable for data clocked into the update stage, active high
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
mode	1 bit	Input	Determines whether data_out is controlled by the boundary scan cell or by the data_in signal
intest	1 bit	Input	INTEST instruction signal
si	1 bit	Input	Serial path from the previous boundary scan cell
data_in	1 bit	Input	Input data from system input pin
data_out	1 bit	Output	Output data
so	1 bit	Output	Serial path to the next boundary scan cell

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1

DW_bc_7

Boundary Scan Cell Type BC_7

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous scan cells with respect to `tck`
- Supports the standard instructions: EXTEST, SAMPLE/ PRELOAD, and BYPASS
- Supports the optional instructions INTEST, RUNBIST, CLAMP, and HIGHZ

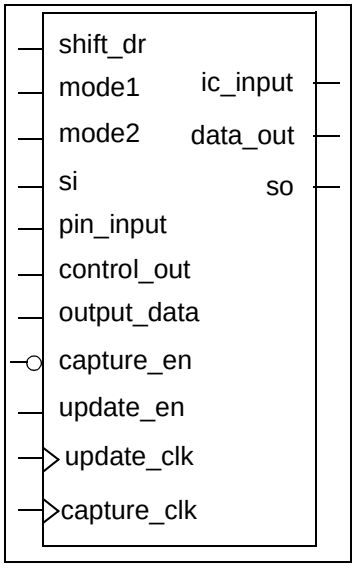
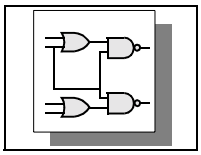


Table 1: Pin Description

Pin Name	Width	Direction	Function
capture_clk	1 bit	Input	Clocks data into the capture stage
update_clk	1 bit	Input	Clocks data into the update stage
capture_en	1 bit	Input	Enable for data clocked into the capture stage, active low
update_en	1 bit	Input	Enable for data clocked into the update stage, active high
shift_dr	1 bit	Input	Enables the boundary scan chain to shift data one stage toward its serial output (tdo)
mode1	1 bit	Input	Determines whether data_out is controlled by the boundary scan cell or by the output_data signal
mode2	1 bit	Input	Determines whether ic_input is controlled by the boundary scan cell or by the pin_input signal
si	1 bit	Input	Serial path from the previous boundary scan cell
pin_input	1 bit	Input	IC system input pin
control_out	1 bit	Input	Control signal for the output enable
output_data	1 bit	Input	IC output logic signal
ic_input	1 bit	Output	IC input logic signal
data_out	1 bit	Output	Output data
so	1 bit	Output	Serial path to the next boundary scan cell

Table 2: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation Test-IEEE-STD-1149-1



Logic – Combinational Overview

The combinational components consist of high-performance logical components. Most components in this category have multiple architectures for each function (architecturally optimized for either performance or area) to provide you with the best architecture for your design goals. All components have a parameterized word length.

DW01_binenc

Binary Encoder

- Parameterized word length
- Inferable using a function call

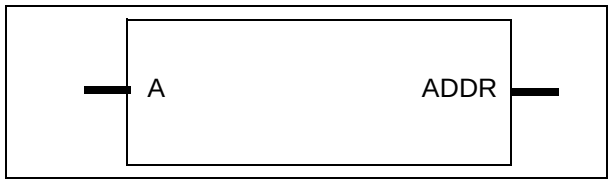


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width*	Input	Input data
ADDR	*ADDR_width*	Output	Binary encoded output data

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of input A
ADDR_width	≥ ceil($\log_2$(*A_width*+1))	Word length of output ADDR

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW01_decode

Decoder

- Parameterized word length
- Inferable using a function call

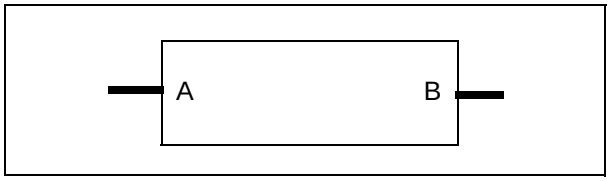


Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width*	Input	Binary input data
B	2^{width}	Output	Decoded output data

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of input A is *width*. Word length of output B is 2^{width}

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW01_mux_any

Universal Multiplexer

- Parameterized word lengths
- Saves coding time by eliminating the need to code muxes explicitly
- Increases design abstraction
- Uses 8-to-1 muxes where possible

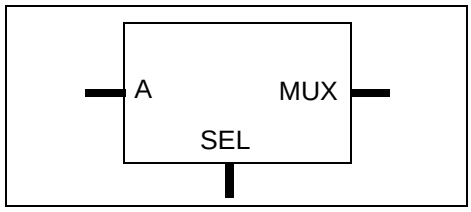


Table 1: Pin Description

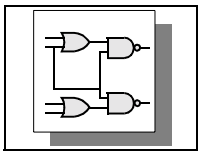
Pin Name	Width	Direction	Function
A	*A_width*	Input	Data input bus
SEL	*SEL_width*	Input	Select input
MUX	*MUX_width*	Output	Multiplexed data out

Table 2: Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
SEL_width	≥ 1	Word length of SEL
MUX_width	≥ 1	A((SEL + 1) × *MUX_width* –1 downto SEL**MUX_width*)

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW01_prienc

Priority Encoder

- Parameterized word length
- Inferable using a function call

Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*A_width*	Input	Input data
INDEX	*INDEX_width*	Output	Binary encoded output data

Table 2: Parameter Description

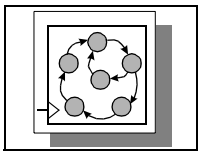
Parameter	Values	Description
A_width	≥ 1	Word length of input A
INDEX_width	≥ ceil($\log_2$[*A_width*+1])	Word length of output INDEX

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

Logic – Sequential Overview

The sequential components consist of high-performance counters, many with either dynamic or static count-to flags. Components in this category have multiple architectures for each function (architecturally optimized for either performance or area) to provide you with the best architecture for your design goals. All components have a parameterized word length.



DW03_bictr_dcnto

Up/Down Binary Counter with Dynamic Count-to Flag

- Parameterized word length
- Terminal count flag for count-to comparison
- Pin-programmable count-to value
- Up/down count control
- Asynchronous reset
- Synchronous counter load
- Synchronous count enable

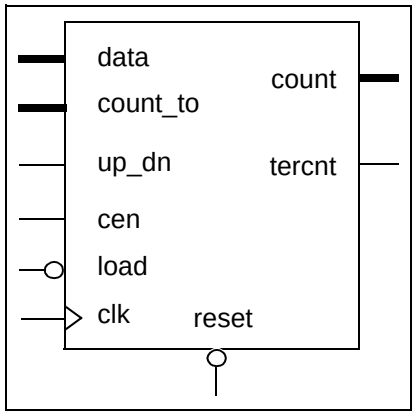


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width*	Input	Counter load input
count_to	*width*	Input	Count compare input
up_dn	1	Input	High for count up and low for count down
load	1	Input	Enable data load to counter, active low
cen	1	Input	Count enable, active high
clk	1	Input	Clock
reset	1	Input	Counter reset, active low
count	*width*	Output	Output count bus
tercnt	1	Output	Terminal count flag, active high

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Width of data input bus

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW03_bictr_scnto

Up/Down Binary Counter with Static Count-to Flag

- Parameterized word length
- Parameterized count-to value
- Up/down count control
- Asynchronous reset
- Loadable count register
- Terminal count flag
- Counter enable

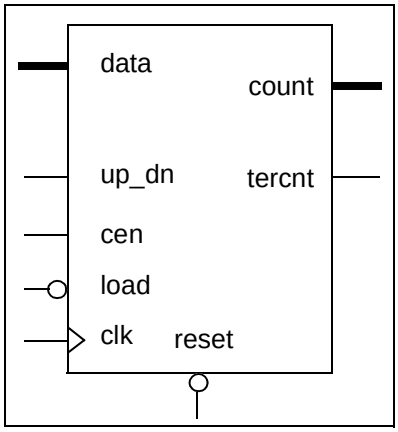


Table 1: Pin Description

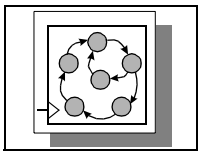
Pin Name	Width	Direction	Function
data	*width*	Input	Counter load input
up_dn	1 bit	Input	High for count up and low for count down
load	1 bit	Input	Enable data load to counter, active low
cen	1 bit	Input	Count enable, active high
clk	1 bit	Input	Clock
reset	1 bit	Input	Counter reset, active low
count	*width*	Output	Output count bus
tercnt	1 bit	Output	Terminal count flag

Table 2: Parameter Description

Parameter	Values	Description
width	1 to 30	Width of data and count
count_to	1 to $2^{width-1}$	Count-to value

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW03_bictr_decode

Up/Down Binary Counter with Output Decode

- Up/down count control
- Asynchronous reset
- Loadable count register
- Counter enable
- Terminal count flag

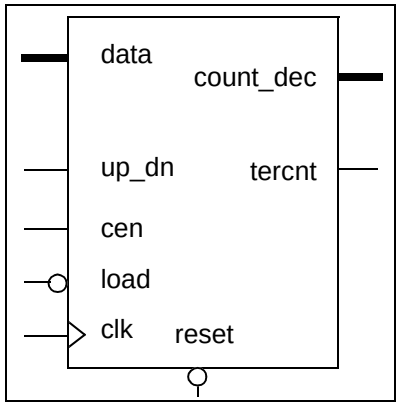


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width*	Input	Counter load input
up_dn	1	Input	High for count up and low for count down
load	1	Input	Enable data load to counter, active low
cen	1	Input	Count enable, active high
clk	1	Input	Clock
reset	1	Input	Counter reset, active low
count_dec	2^{width}	Output	Binary decoded count value
tercnt	1	Output	Terminal count flag

Table 2: Parameter Description

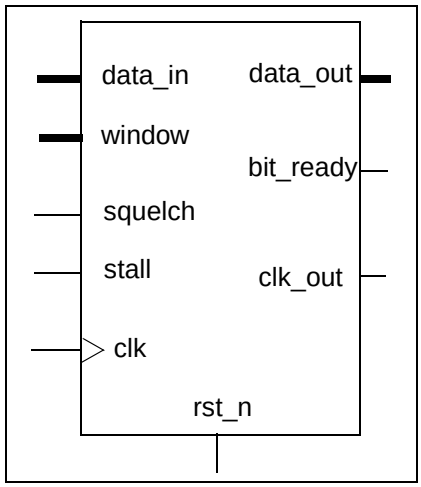
Parameter	Values	Function
width	≥ 1	Width of data input bus

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW_dpll_sd

Digital Phase Locked Loop



- Parameterizable divisor (ratio of reference clock to baud rate)
- Multichannel data recovery (recovery of channels that accompany the locked channel)
- Stall input for power saving mode and/or prescaler (allowing one DW_dpll_sd to recover data at multiple rates)
- Squelch input for ignoring phase information when channel data is unknown or unconnected
- Sampling window control to aid data recovery under harsh conditions
- Parameterizable gain selection to meet a variety of application needs
- Parameterizable filter (controls phase correction reactiveness from minor phase errors)

Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Reference clock
rst_n	1 bit	Input	Asynchronous reset, active low
stall	1 bit	Input	Stalls everything except synchronizer, active high
squelch	1 bit	Input	Turns off phase detection. When high no phase correction is carried out leaving DPLL free running, active high
window	ceil(log2(*windows*))	Input	Sampling window selector[a]
data_in	*width* bit(s)	Input	Serial input data stream
clk_out	1 bit	Output	Recovered Clock
bit_ready	1 bit	Output	Output data ready flag
data_out	*width* bit(s)	Output	Recovered output data stream

a. The minimum value must be 1.

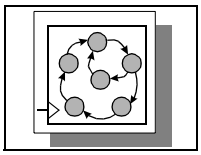


Table 2: Parameter Description

Parameter	Values	Description
width	1 to 16 Default: 1	Number of input serial channels
divisor	4 to 256 Default: 4	Determines the number of samples per input clock cycle
gain	1 to 2 Default: 1	Phase correction factor for the absolute value of clock phase error greater than |1| 1 = 50% phase correction 2 = 100% phase correction
filter	0 to 8 Default: 2	Phase correction control for +/- 1 clock phase error region. 0 = no correction 1 = always correct For integer N > 1, correct after N samples at a current phase (such as, N consecutive samples at +1 or N consecutive samples at -1)
windows	1 to (divisor+1)/2 Default: 1	Number of sampling windows for the input serial data stream

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple synthesis model	DesignWare-Foundation
cla	Carry look-ahead architecture synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

DW03_lfsr_dcnto

LFSR Counter with Dynamic Count-to Flag

- Dynamically programmable count-to value that indicates when the counter reaches a specified value
- High speed, area-efficient
- Asynchronous reset
- Terminal count

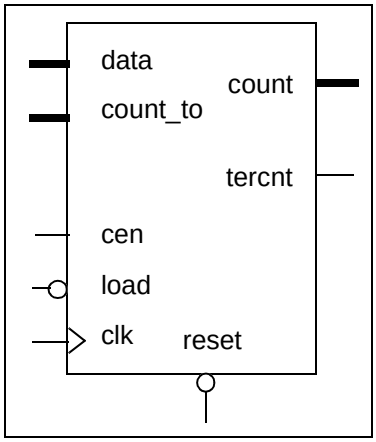


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width* bit(s)	Input	Input data
count_to	*width* bit(s)	Input	Input count_to_bus
load	1 bit	Input	Input load data to counter, active low
cen	1 bit	Input	Input count enable
clk	1 bit	Input	Clock
reset	1 bit	Input	Asynchronous reset, active low
count	*width* bit(s)	Output	Output count bus
tercnt	1 bit	Output	Output terminal count

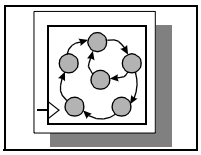
Table 2: Parameter Description

Parameter	Legal Range[a]	Description
width	1 to 50	Word length of counter

a. The upper bound of the legal range is a guideline to ensure reasonable compile times

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW03_lfsr_scnto

LFSR Counter with Static Count-to Flag

- Parameterized count-to value to indicate when the counter reaches a specified value
- Parameterized word length
- High speed, area-efficient
- Asynchronous reset
- Terminal count flag

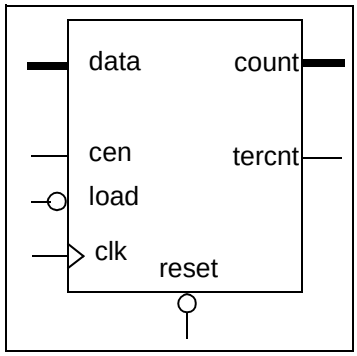


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width* bit(s)	Input	Input data
load	1 bit	Input	Input load, active low
cen	1 bit	Input	Input count enable
clk	1 bit	Input	Clock
reset	1	Input	Asynchronous reset, active low
count	*width* bit(s)	Output	Output count bus
tercnt	1 bit	Output	Output terminal count

Table 2: Parameter Description

Parameter	Values[a]	Function
width	2 to 50	Word length of counter
count_to	1 to $2^{width-2}$	count_to bus

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW03_lfsr_load

LFSR Counter with Loadable Input

- Parameterized word length
- Loadable counter registers
- High speed, area-efficient
- Asynchronous reset
- Terminal count

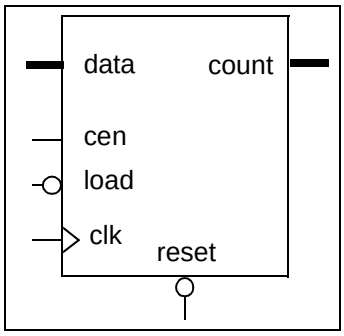


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width* bit(s)	Input	Input data
load	1 bit	Input	Input load data to counter, active low
cen	1 bit	Input	Input count enable
clk	1 bit	Input	Clock
reset	1 bit	Input	Asynchronous reset, active low
count	*width* bit(s)	Output	Output count bus

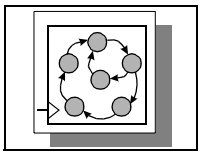
Table 2: Parameter Description

Parameter	Values[a]	Description
width	1 to 50	Word length of counter

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



DW03_lfsr_updn

LFSR Up/Down Counter

- High speed, area-efficient
- Pseudorandom sequence generator
- Up/down count control
- Asynchronous reset
- Terminal count flag

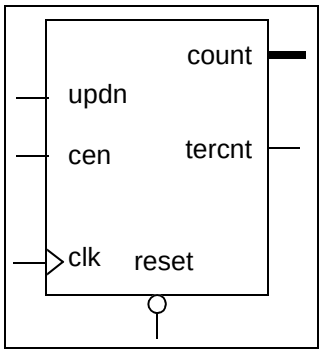


Table 1: Pin Description

Pin Name	Width	Direction	Function
updn	1 bit	Input	Input high for count up and low for count down
cen	1 bit	Input	Input count enable
clk	1 bit	Input	Clock
reset	1 bit	Input	Asynchronous reset, active low
count	*width* bit(s)	Output	Output count bus
tercnt	1 bit	Output	Output terminal count

Table 2: Parameter Description

Parameter	Values[a]	Description
width	2 to 50	Word length of counter

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation

DW03_updn_ctr

Up/Down Counter

- Up/down count control
- Asynchronous reset
- Loadable count register
- Counter enable
- Terminal count flag
- Multiple synthesis implementations

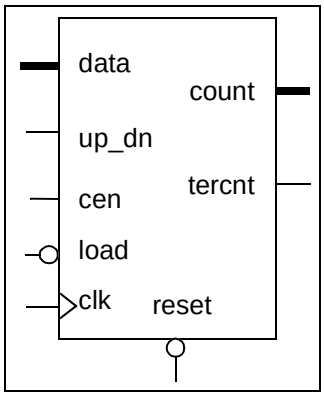


Table 1: Pin Description

Pin Name	Width	Direction	Function
data	*width* bit(s)	Input	Input data bus
up_dn	1 bit	Input	Count up (up_dn = 1) or count down (up_dn = 0)
load	1 bit	Input	Counter load enable, active low
cen	1 bit	Input	Counter enable, active high
clk	1 bit	Input	Clock
reset	1 bit	Input	Asynchronous counter reset, active low
count	*width* bit(s)	Output	Output count bus
tercnt	1 bit	Output	Terminal count flag

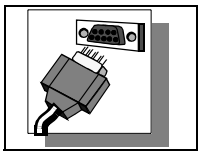
Table 2: Parameter Description

Parameter	Value	Function
width	≥ 1	Width of count output bus

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
rpl	Ripple carry synthesis model	DesignWare-Foundation
cla	Carry look-ahead synthesis model	DesignWare-Foundation
clf	Fast carry look-ahead synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



Application Specific – Interface Overview

The Interface IP consist of the DW_debugger IP.

DW_debugger
On-Chip ASCII Debugger

- Low gate count
- Parameterized data widths

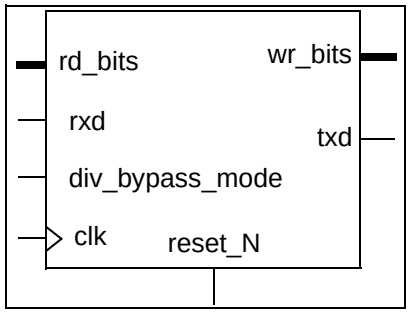


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
reset_N	1 bit	Input	Synchronous reset, active low
rd_bits	*rd_bits_width* bit(s)	Input	Input data bus
rxd	1 bit	Input	Receive data
wr_bits	*wr_bits_width* bit(s)	Output	Output data bus
txd	1 bit	Output	Transmit data
div_bypass_mode	1 bit	Input	Clock Divider Bypass Control, active high

Table 2: Parameter Description

Parameter	Values	Description
rd_bits_width	8 to 2048 Default: 8	Width of rd_bits
wr_bits_width	8 to 2048 Default: 8	Width of wr_bits
clk_freq	≥1 (must be a whole number) Default: 1	Clock rate in MHz
baud_rate	300, 600, 1200, 2400, 4800, 9600, or 19200 Default: 19200	Sets the baud rate of the UART
mark_parity	0 or 1 Default: 1	Sets the fixed value of the parity bit from the UART transmitter

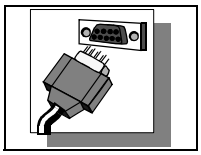
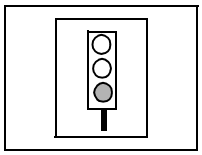


Table 3: Synthesis Implementations

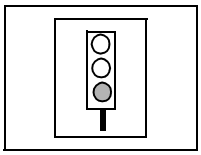
Implementation Name	Function	License Required
str	Synthesis model	DesignWare-Foundation



Application Specific – Control Logic

The Control Logic IP, consist of a family of arbiters. DesignWare Building Block IP of the arbiter family are distinguished from each other primarily by the arbitration scheme they embody.

The IP DW_arbiter_sp and DW_arbiter_dp are based on the static fixed priority scheme and dynamically programmable priority scheme, respectively. Each of these IP has multiple architectural implementations optimized for timing or area. The number of clients connected to the arbiter is parametrizable from 2 to 32. Other features like parking and locking are available through parameter selection.



DW_arbiter_2t

Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs

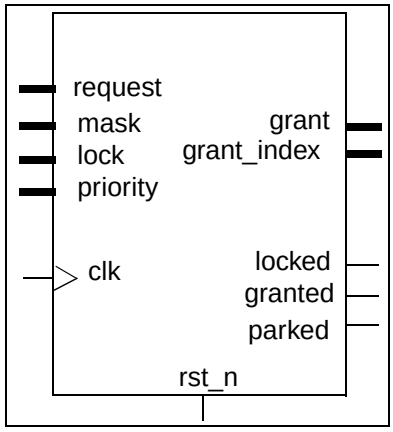


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Active low input reset
request	*n* bit(s)	Input	Input request from clients
priority	$n \times p_width$ bit(s)	Input	Priority vector from the clients of the arbiter
lock	*n* bit(s)	Input	Active high signal to lock the grant to the current request. By setting lock(*i*) = 1, the arbiter is locked to the request (*i*) if it is currently granted. For lock (*i*) = 0, the lock on the arbiter is removed.
mask	*n* bit(s)	Input	Active high input to mask specific clients. By setting mask(*i*) = 1, request(*i*) is masked. For mask(*i*) = 0, the mask on the request(*i*) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to park_index
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
locked	1 bit	Output	Flags that the arbiter is locked by a client
grant	*n* bit(s)	Output	Grant output
grant_index	ceil($\log_2 n$) bit(s)	Output	Index of the requesting client that has been currently granted or the client designated by park_index in park_mode

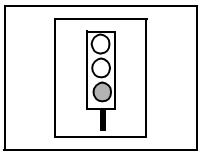
Table 2: Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
p_width	1 to 5 Default: 2	Width of the priority vector of each client
park_mode	0 or 1 Default: 1	park mode = 1 includes logic to enable parking when no clients are requesting and park_mode = 0 contains no logic for parking.
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	output_mode = 1 includes registers at the outputs output_mode = 0 contains no output registers

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
cla	Carry-look-ahead synthesis model	DesignWare-Foundation
clas	Carry-look-ahead/select synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_arbiter_dp

Arbiter with Dynamic Priority Scheme

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs

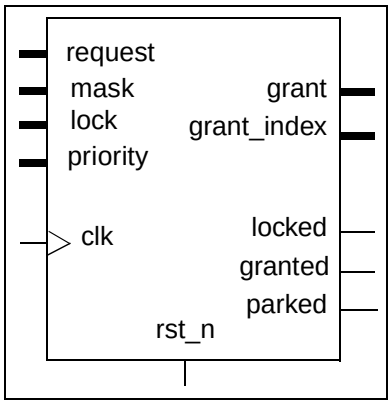


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Input reset, active low
request	*n* bit(s)	Input	Input request from clients
priority	*n**ceil($\log_2 n$) bit(s)	Input	Priority vector from the clients of the arbiter
lock	*n* bit(s)	Input	Signal to lock the grant to the current request. By setting lock (*i*) = 1, the arbiter is locked to the request (*i*) if it is currently granted. For lock (*i*) = 0 the lock on the arbiter is removed.
mask	*n* bit(s)	Input	Input to mask specific clients. By setting mask (*i*) = 1, request (*i*) is masked. For mask (i) = 0 the mask on the request (i) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to client designated by park_index
granted	1 bit	Output	Flag to indicate that the arbiter has issued a grant to one of the requesting clients
locked	1 bit	Output	Flag to indicate that the arbiter is locked by a client
grant	*n* bit(s)	Output	Grant output
grant_index	$\log_2 n$ bit(s)	Output	Index of the requesting client that has been currently granted or the client designated by park_index in park_mode

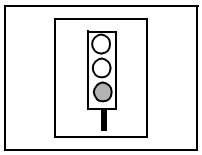


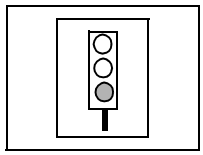
Table 2: Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
park_mode	0 or 1 Default: 1	park mode = 1 includes logic to enable parking when no clients are requesting and park_mode = 0 contains no logic for parking.
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	output_mode = 1 includes registers at the outputs output_mode = 0 contains no output registers

Table 3: Synthesis Implementations [a]

Implementation Name	Function	License Required
cla	Carry-look-ahead synthesis model	DesignWare-Foundation
clas	Carry-look-ahead/select synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_arbiter_fcfs

Arbiter with First-Come-First-Served Priority Scheme

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs

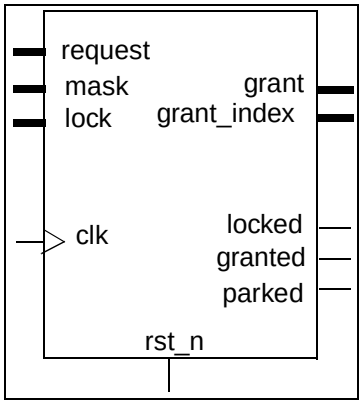


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Active low input reset
request	*n* bit(s)	Input	Input request from clients
lock	*n* bit(s)	Input	Active high signal to lock the grant to the current request. By setting lock(*i*) = 1, the arbiter is locked to the request (*i*) if it is currently granted. For lock (*i*) = 0, the lock on the arbiter is removed.
mask	*n* bit(s)	Input	Active high input to mask specific clients. By setting mask(*i*) = 1, request(*i*) is masked. For mask(*i*) = 0, the mask on the request(*i*) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to park_index
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
locked	1 bit	Output	Flags that the arbiter is locked by a client
grant	*n* bit(s)	Output	Grant output
grant_index	ceil($\log_2 n$) bit(s)	Output	Index of the requesting client that has been currently granted or the client designated by park_index in park_mode

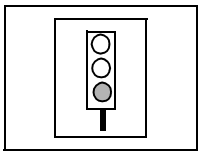
Table 2: Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
park_mode	0 or 1 Default: 1	park mode = 1 includes logic to enable parking when no clients are requesting and park_mode = 0 contains no logic for parking.
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	output_mode = 1 includes registers at the outputs output_mode = 0 contains no output registers

Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Required
cla	Carry-look-ahead synthesis model	DesignWare-Foundation
clas	Carry-look-ahead/select synthesis model	DesignWare-Foundation

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.



DW_arbiter_sp

Arbiter with Static Priority Scheme

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant to a client when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs

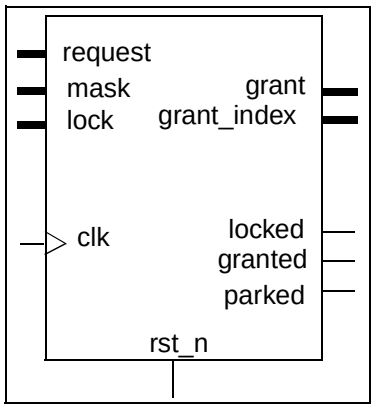


Table 1: Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Active low input reset
request	*n* bit(s)	Input	Input request from clients
lock	*n* bit(s)	Input	Active high signal to lock input. By setting lock(*i*) = 1, the arbiter is locked to the request (*i*) if it is currently granted. For lock (*i*) = 0, the lock on the arbiter is removed.
mask	*n* bit(s)	Input	Active high input to mask specific clients. By setting mask(*i*) = 1, request(*i*) is masked. For mask(*i*) = 0, the mask on the request(*i*) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to park_index
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
locked	1 bit	Output	Flags that the arbiter is locked by a client
grant	*n* bit(s)	Output	Grant output
grant_index	$\log_2 n$ bit(s)	Output	Index of the requesting client that has been currently issued the grant or the client designated by park_index in park_mode

Table 2: Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
park_mode	0 or 1 Default: 1	park mode = 1 includes logic to enable parking when no clients are requesting and park_mode = 0 contains no logic for parking.
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	output_mode = 1 includes registers at the outputs output_mode = 0 contains no output registers

Table 3: Synthesis Implementations

Implementation Name	Function	License Required
rpl	Ripple synthesis model	DesignWare-Foundation
cla	Carry-look-ahead synthesis model	DesignWare-Foundation

DSP Library Overview

The DesignWare DSP Library consists of two digital FIR filter components designed for applications requiring programmable coefficients, and either high-speed or area-efficient filtering.

DW07_fir100 is a high-speed FIR Filter with a word-parallel, transposed canonical architecture. It processes one sample per clock cycle. Because of its inherent pipelining, the clock rate is essentially independent of the filter order, allowing for the implementation of very large order filters. It is also easily cascaded to help partition large order filters.

DW07_fir110 is an FIR Filter processor with a word-serial architecture that is area-efficient and designed for low to medium sampling rates. It processes one filter tap per clock cycle. An N^{th} order filter, therefore, processes one sample every N clock cycles. The processor consists of a finite state machine, coefficient and data memory, and an arithmetic datapath.

Both components have parameterized data, coefficient and accumulator word lengths, parameterized filter order, and serially loadable coefficients.

For more information about the DSP IP, refer to the *DesignWare DSP and GTECH Libraries Databook*.

GTECH Library Overview

Synopsys provides the GTECH technology-independent library to aid users in developing technology-independent parts. Also, DesignWare IP often use these cells for their implementation. This generic technology library, called gtech.db, contains common logic elements. gtech.db can be found under the Synopsys root directory in libraries/syn. Simulation models are located under the Synopsys root directory in packages/gtech/src (VHDL) and packages/gtech/src_ver (Verilog).

For more information about the GTECH IP, refer to the *DesignWare DSP and GTECH Libraries Databook*.

AMBA On-Chip Bus Logic and Peripherals IP

AMBA® is a standard bus architecture system developed by ARM® for rapid development of processor-driven systems. AMBA also allows a number of bus peripherals and resources to be connected in a consistent way. The following Synopsys AMBA 2.0-compliant components are briefly described in this section:

Table 1: Alphabetical List of the AMBA Synthesizable IP

Name and Page	Description
DW_ahb, page 232	Advanced High-performance Bus (AHB)
DW_ahb_h2h, page 234	AHB to AHB Bridge
DW_ahb_icm, page 235	AMBA AHB Multi-layer Interconnection Matrix
DW_ahb_ictl, page 236	AHB Interrupt Controller
DW_apb, page 237	Advanced Peripheral Bus (APB)
DW_apb_gpio, page 238	General Purpose Programmable I/O
DW_apb_ictl, page 240	APB Interrupt Controller
DW_apb_i2c, page 239	APB I^2C Interface
DW_apb_rap, page 241	Remap and Pause
DW_apb_rtc, page 242	APB Real Time Clock
DW_apb_ssi, page 243	APB Synchronous Serial Interface
DW_apb_timers, page 245	Programmable Timers
DW_apb_uart, page 246	Universal Asynchronous Receiver/Transmitter
DW_apb_wdt, page 248	APB Watch Dog Timer

A brief introduction to the AMBA On-Chip Bus can be found at the following location:

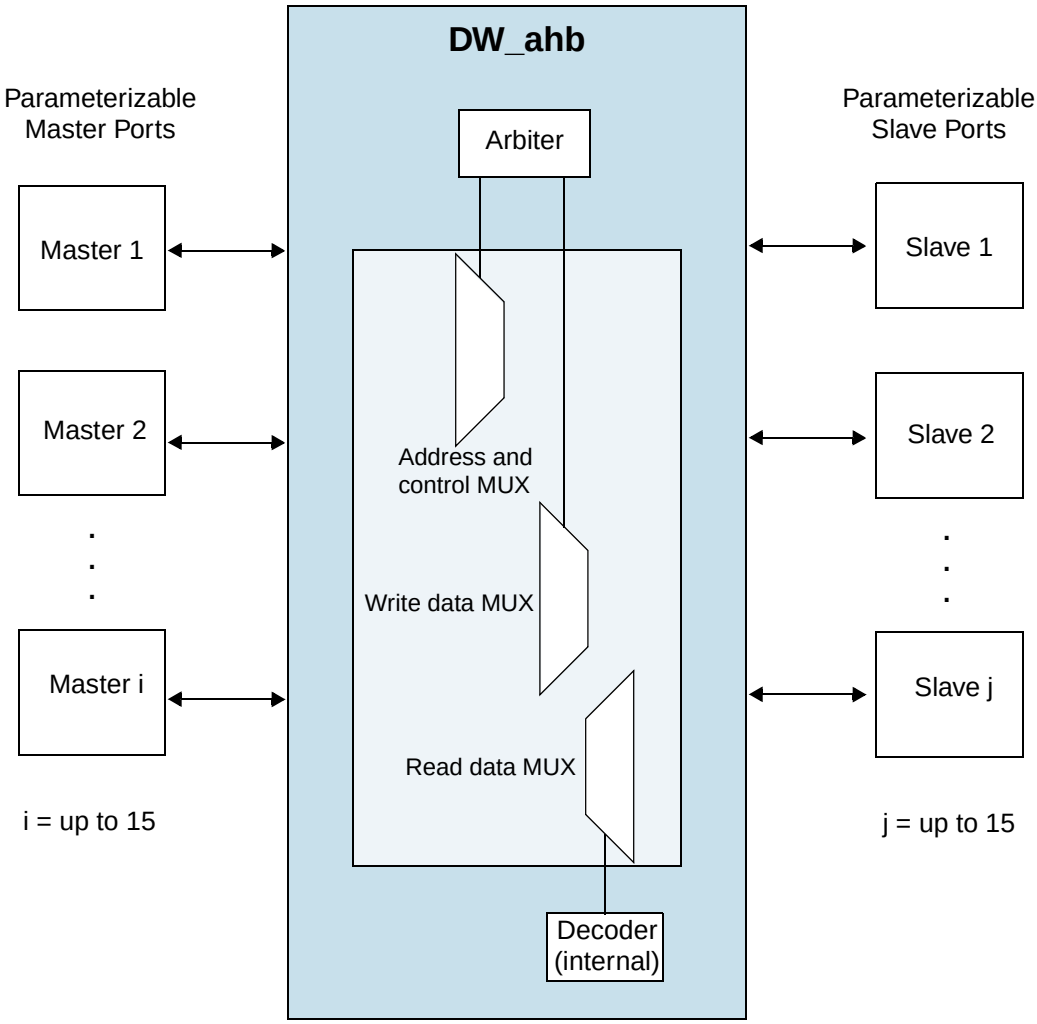
http://www.synopsys.com/products/designware/dw_amba.html

AMBA Compliant

DW_ahb

Advanced High-performance Bus

- Configuration of AMBA Lite system
- Configuration of up to 15 masters in a non-AMBA Lite system
- Configuration of up to 15 slaves in a non-AMBA Lite system
- Configuration of data bus width of up to 256 bits
- Configurable system address width of 32 or 64 bits
- Configuration of system endianness — Big or Little Endian
- Optional Arbiter Slave Interface
- Programmable arbitration scheme:
- Weighted Token
- Programmable or Fixed priority
- Fair-Among-Equals
- Arbitration for up to 15 masters
- Optional Bus decoder
- External debug mode signal
- Individual grant signals for each master
- Support for split, burst, and locked transfers
- Optional support for early burst termination
- Configurable support for termination of undefined length bursts by masters of equal or higher priority
- Configurable or hardcoded priority assignments to masters
- Disabling of masters and protection against self disable
- Optional support for AMBA memory remap feature
- Optional support for pausing of the system
- Contiguous and non-contiguous memory allocation options for slaves
- Component version ID register



The *DesignWare DW_ahb Databook* is available at:

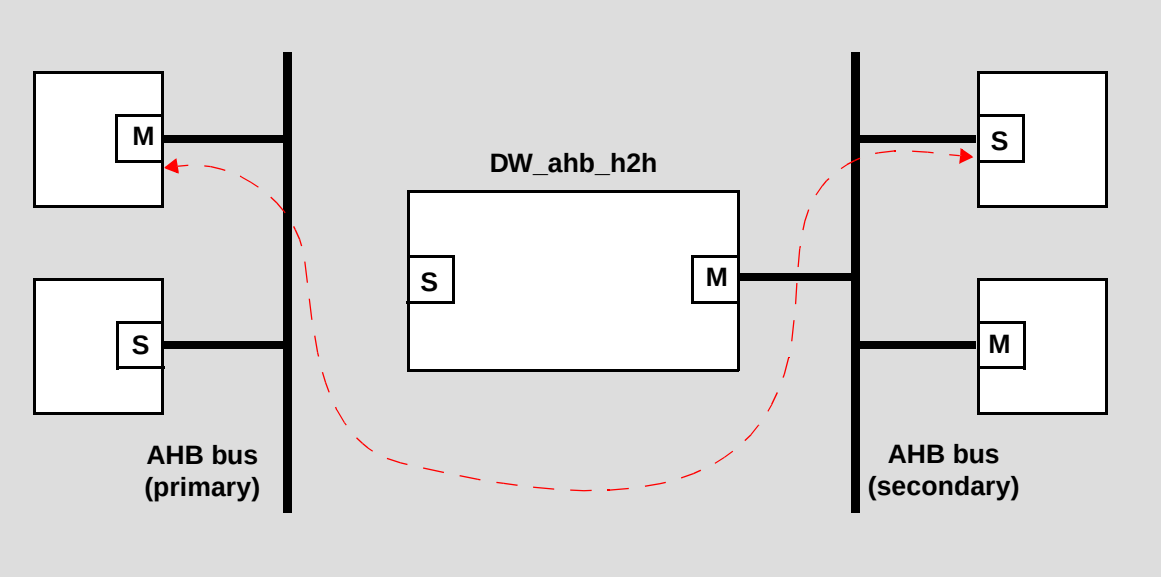
http://www.synopsys.com/products/designware/docs

AMBA Compliant

DW_ahb_h2h

AMBA AHB to AHB Bridge

- Asynchronous clocks, any clock ratio
- Synchronous clocks, any integer clock ratio
- Any address bus width, data bus width, any endianness
- No split capable slave
- Bursts forwarded as single NSEQ beats
- Self timed1 low gate count implementation
- Internal timeout counters for deadlock protection
- Synchronizers bypass selection pin



The *DesignWare DW_ahb_h2h Databook* is available at:

http://www.synopsys.com/products/designware/docs

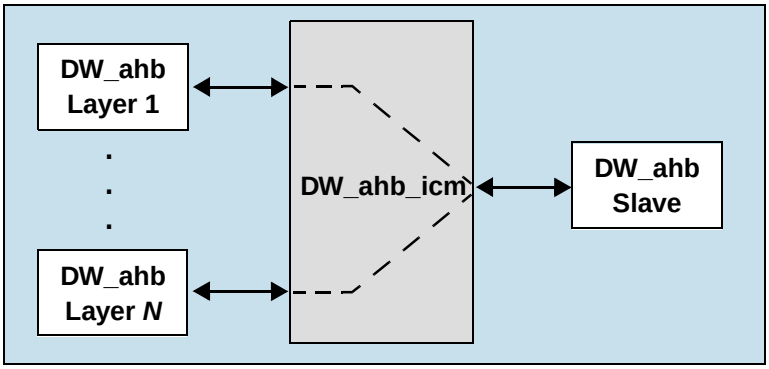
DW_ahb_icm

AMBA AHB Multi-layer Interconnection Matrix

- Layer arbitration and master multiplexing
- Input stage address and control holding registers for each layer
- Mapping of slave response onto correct layer
- Returning of splits onto the correct layer
- Common clock and reset shared amongst all layers

Some uses of the DW_ahb_icm are:

- An add-on interface for AHB slaves to allow them to be used in a multi-layer system.
- Support for an AHB master to operate in an AMBA Lite mode within a multi-layer system.



The *DesignWare DW_ahb_icm Databook* is available at:

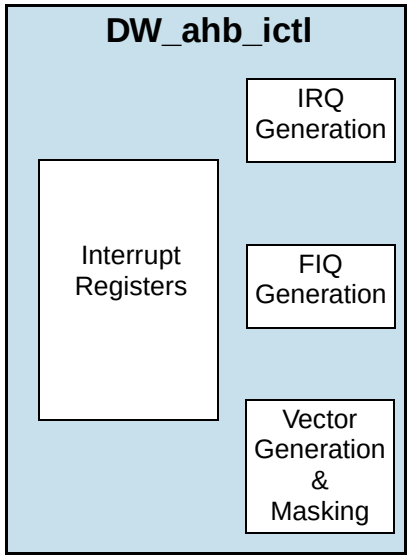
http://www.synopsys.com/products/designware/docs

AMBA Compliant

DW_ahb_ictl

AHB Interrupt Controller

- 2 to 64 IRQ normal interrupt sources
- 1 to 8 FIQ fast interrupt sources (optional)
- Vectored interrupts (optional)
- Software interrupts
- Priority filtering (optional)
- Masking
- Optional scan mode

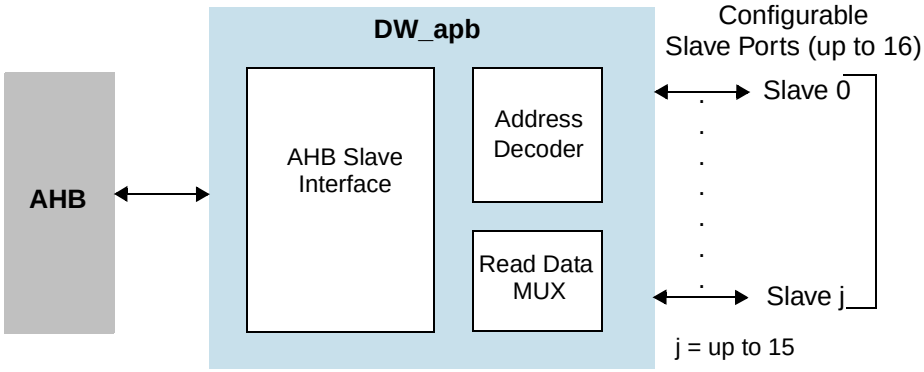


The *DesignWare DW_ahb_ictl Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_apb
Advanced Peripheral Bus

- Incorporates APB Bridge and APB bus functionality
- AHB Slave
- Support for up to 16 APB slaves
- Supports synchronous hclk/pclk; hclk is an integer multiple of pclk
- Supports Little Endian APB slaves
- Supports 32, 64, 128, 256 AHB data buses
- Support for single and burst AHB transfers



The *DesignWare DW_apb Databook* is available at:

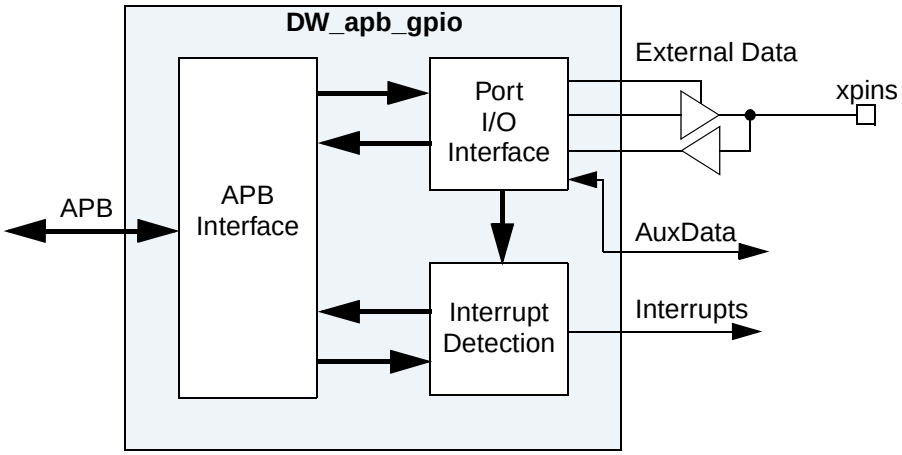
http://www.synopsys.com/products/designware/docs

AMBA Compliant

DW_apb_gpio

General Purpose Programmable I/O

- APB interface to/from APB Bridge
- Up to 128 independently configurable pins
- Up to four ports, A to D, which are separately configurable
- Port A can be configured to act in interrupt mode
- Configurable hardware and software control for each part
- GPIO ID
- Separate data registers and data direction registers for each port
- Separate auxiliary data input, data output, and data control for each I/O in Hardware Control mode
- Configurable debounce logic with an external slow clock to debounce interrupts
- Option to generate single or multiple interrupts
- Component version ID register



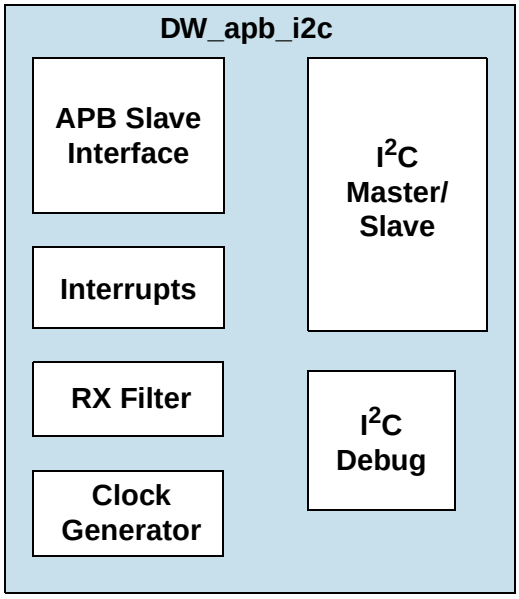
The *DesignWare DW_apb_gpio Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_apb_i2c

APB I^2C Interface

- Two-wire I^2C serial interface
- Three speeds:
  - Standard mode (100 Kb/s)
  - Fast mode (400 Kb/s)
  - High speed mode (3.4 Mb/s)
- Master or slave I^2C operation
- 7- or 10-bit addressing
- Component version ID register
- Ignores CBUS addresses (an older ancestor of I^2C that used to share the I^2C bus)
- Transmit and receive buffers
- Interrupt or polled mode operation
- Simple software interface consistent with DesignWare APB peripherals
- Digital filter for the received SDA and SCL lines



The *DesignWare DW_apb_i2c Databook* is available at:

http://www.synopsys.com/products/designware/docs

AMBA Compliant

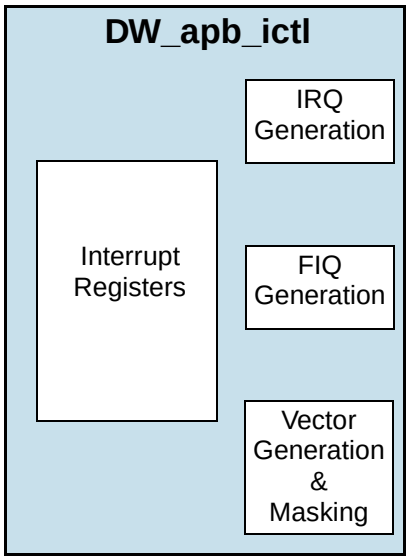
DW_apb_ictl

APB Interrupt Controller

- 2 to 64 IRQ normal interrupt sources
- 1 to 8 FIQ fast interrupt sources (optional)
- Vectored interrupts (optional)
- Software interrupts
- Priority filtering (optional)
- Masking
- Scan mode
- Component version ID register
- Component version ID register

Note

DW_apb_ictl is an exact replacement for the original component DW_amba_ictl (name change only).



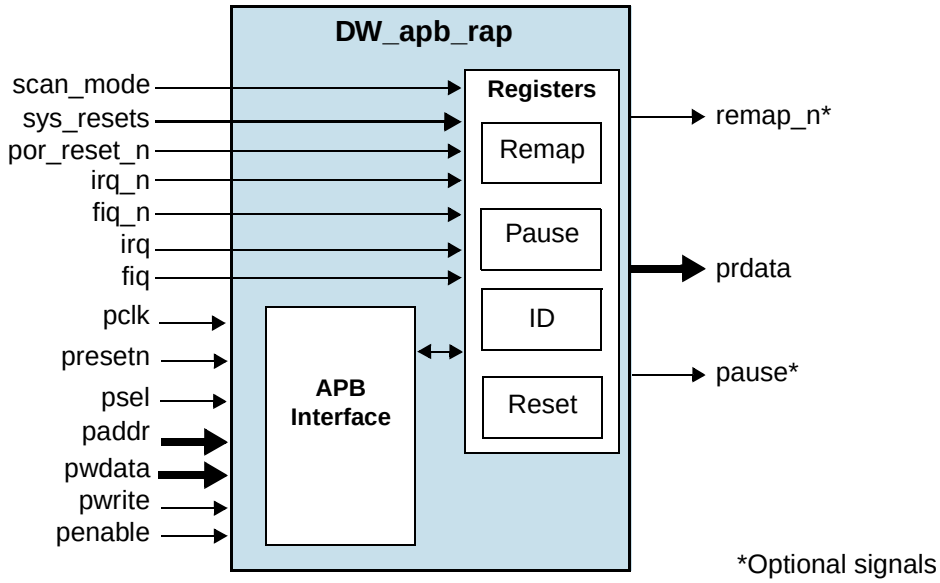
The *DesignWare DW_apb_ictl Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_apb_rap

AMBA APB Remap and Pause

- Remap Control switches DW_ahb address decoder from boot mode to normal mode operation
- Identification code register implements configurable, read-only ID register for accessing user-defined values
- Pause mode puts DW_ahb arbiter into low-power (pause) mode
- Reset Status Register keeps track of status from up to eight separate system reset signals
- Component version ID register



The *DesignWare DW_apb_rap Databook* is available at:

http://www.synopsys.com/products/designware/docs

AMBA Compliant

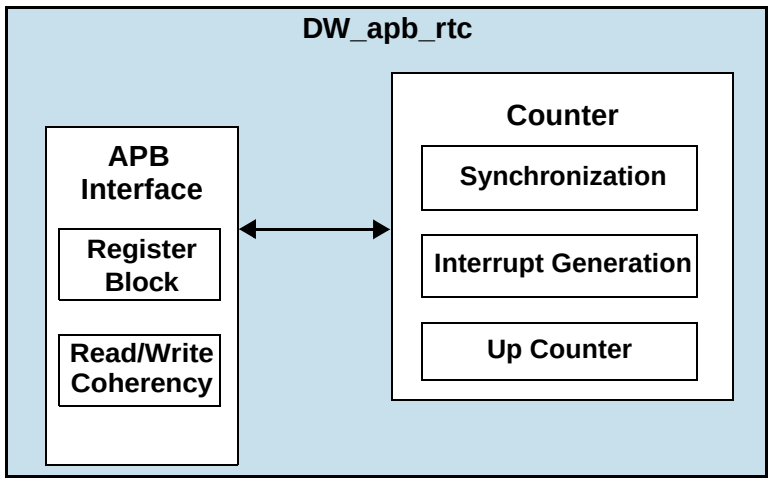
DW_apb_rtc

AMBA APB Real Time Clock

- APB slave interface with read/write coherency for registers
- Incrementing counter and comparator for interrupt generation
- User-defined parameters:
  - APB Data Bus Width
  - Counter Width
  - Clock relationship between bus clock and counter clock
  - Interrupt polarity level
  - Interrupt clock domain location
  - Counter enable mode

Some uses of the DW_apb_rtc are:

- Real-time clock – used with software for keeping track of time.
- Long-term, exact chronometer – When clocked with a 1 Hz clock, it can keep track of time from now up to 136 years in the future.
- Alarm function – generates an interrupt after a programmed number of cycles.
- Long-time, base counter – clocked with a very slow clock signal.
- Counter wrap mode



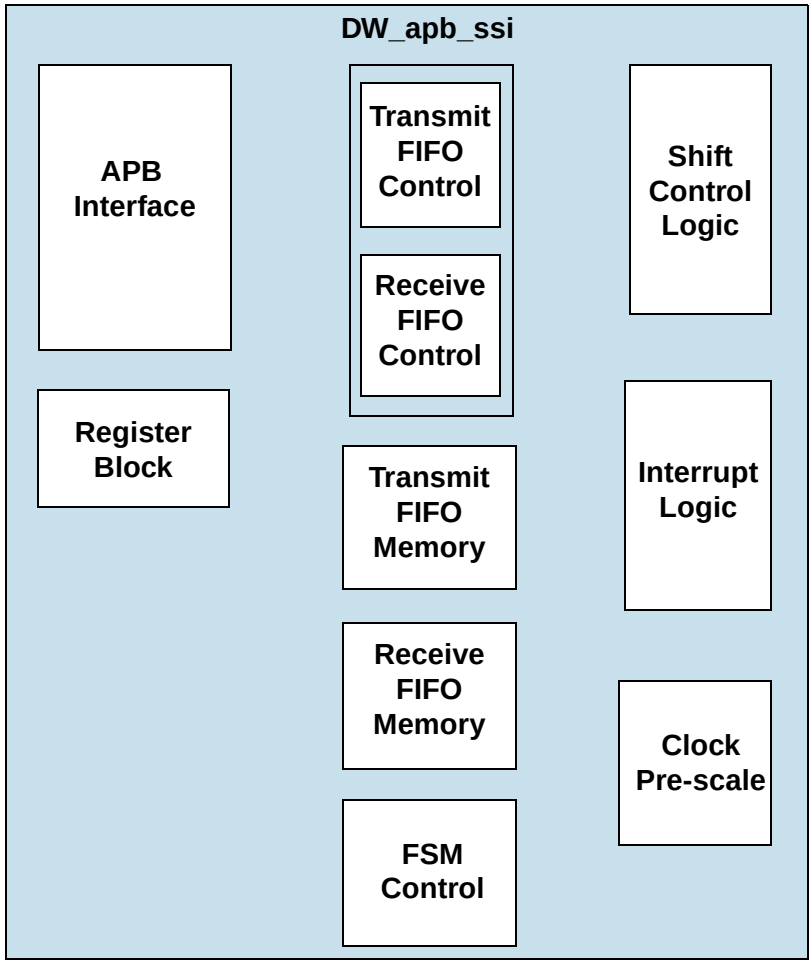
The *DesignWare DW_apb_rtc Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_apb_ssi

AMBA APB Synchronous Serial Interface

- AMBA APB interface – Allows for easy integration into an AMBA SoC implementation.
- Scalable APB data bus width – APB data bus widths of 8, 16 and 32 are supported.
- Serial Master or serial slave operation – Enables serial communication with serial master or serial slave peripheral devices.
- Independent masking of interrupts – Master collision, transmit FIFO overflow, transmit FIFO empty, receive FIFO full, receive FIFO underflow, and receive FIFO overflow interrupts can all be masked independently.
- Multi-master contention detection – Informs the processor of multiple serial master accesses on the serial bus.
- Bypass of meta-stability flip-flops for synchronous clocks – When the APB clock (pclk) and the DW_apb_ssi serial clock (ssi_clk) are synchronous, meta-stable flip-flops are not used when bringing control signals across these clock domains.
- Component version ID register
- Programmable features:
  - Serial interface operation – User can choose from Motorola SPI, Texas Instruments Synchronous Serial Protocol or National Semiconductor Microwire.
  - Clock bit-rate – User can dynamically control the serial bit rate of the data transfer. Used only in serial master mode of operation.
  - Data frame size (4 to 16 bits) – The frame size of each data transfer is under the control of the programmer.
- Configurable features:
  - FIFO depth – User can configure the depth of the transmit and receive FIFO buffers to 1 to 256 words deep. The FIFO width is fixed at 16 bits.
  - Number of slave select outputs – When operating as a serial master 1 to 16 serial slave select output signals can be generated.
  - Hardware/software slave-select – Dedicated hardware slave-select lines can be used or software control can be used to target the serial slave device.
  - Combined or individual interrupt lines – The user may choose to bring all individual interrupt lines or one combined interrupt line from the MacroCell to the interrupt controller.
  - Interrupt polarity – This configuration option selects the serial clock phase of the SPI format directly after reset.



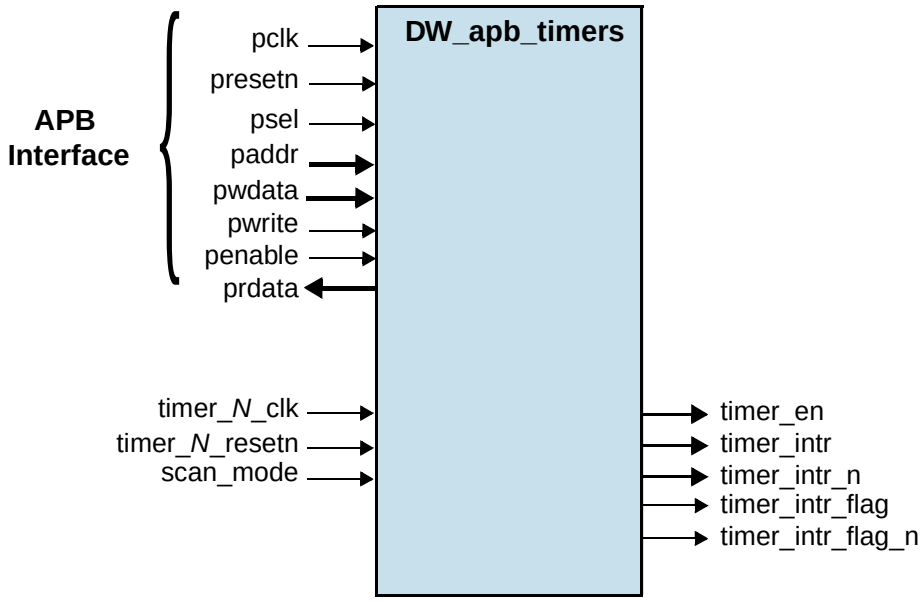
The *DesignWare DW_apb_ssi Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_apb_timers

AMBA APB Programmable Timers

- Up to eight programmable timers
- Configurable timer width: 8 to 32 bits
- Support for two operation modes: free running and user-defined count
- Support for independent clocking of timers
- Configurable polarity for each individual interrupt
- Configurable option for a single or combined interrupt output flag
- Configurable option to have read/write coherency registers for each timer
- Configurable option to include timer toggle output, which toggles each time counter reloads
- Component version ID register



The *DesignWare DW_apb_timers Databook* is available at:

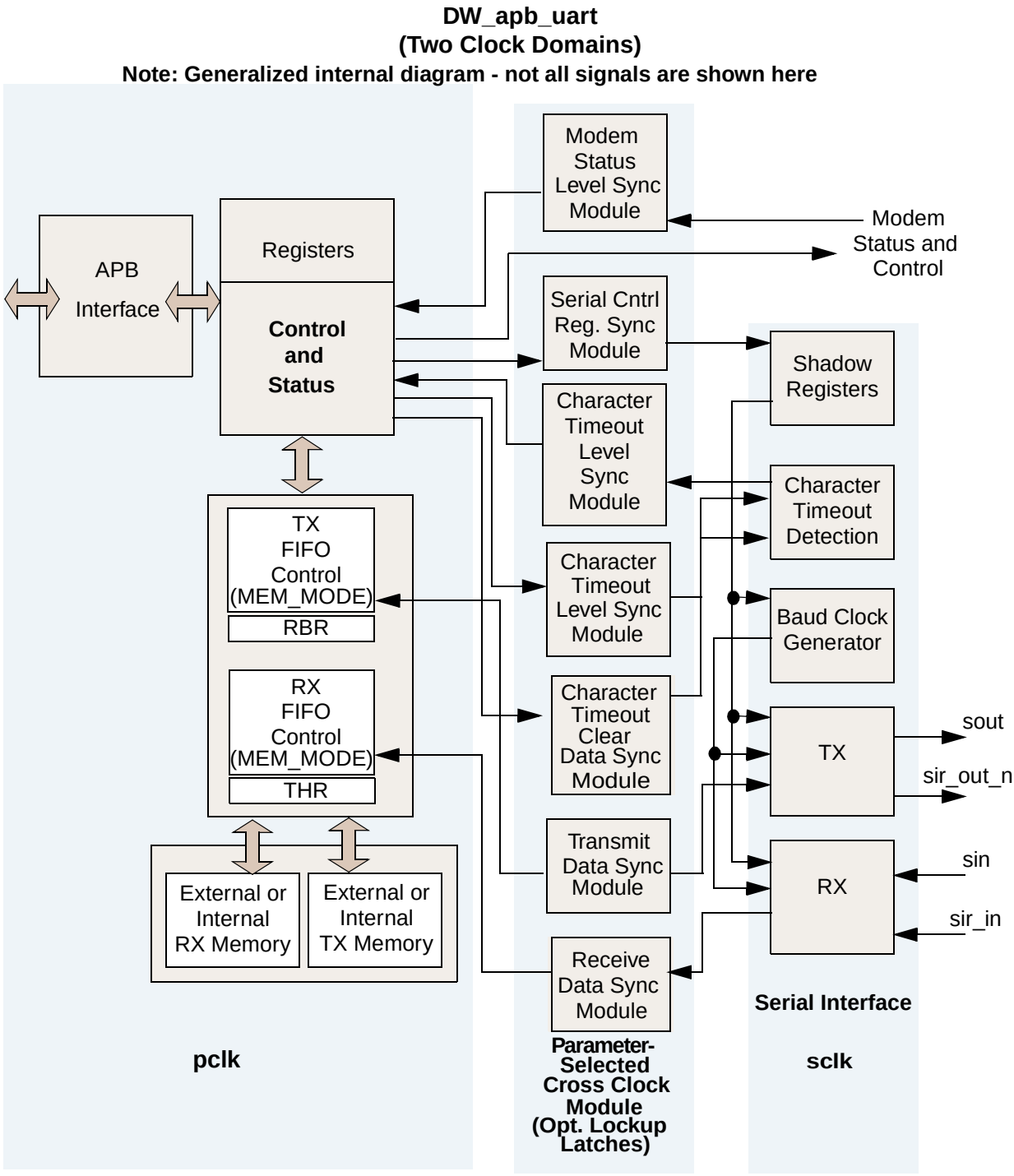
http://www.synopsys.com/products/designware/docs

AMBA Compliant

DW_apb_uart

AMBA APB Universal Asynchronous Receiver/Transmitter

- Functionality is based on the industry standard 16550
- AMBA 2.0 compliant APB Interface with synthesis selectable prdata & pwdata bus widths (8, 16, 32)
- Synthesis selectable IrDA 1.0 SIR Mode support with up to 115.2 Kbaud data rate.
- Programmable FIFO disabling.
- Modem and status lines are independently controlled
- External memory read enable signals for RAM wake-up when external RAMs are selected.
- Synthesis selectable transmit and receive FIFO depths
- Synthesis selectable internal DesignWare D-flip-flop based RAM
- Programmable FIFO disabling
- Synthesis selectable Programmable Transmitter Holding Register (THRE) Interrupt Mode.
- Synthesis selectable synchronous or asynchronous external Read Port RAM interface when external RAMs are selected
- Synthesis selectable asynchronous serial clock support (pclk or pclk and sclk)
- Synthesis selectable ‘lock-up latch’ insertion before clock boundary crossing, in two-clock implementations, for test purposes
- Synthesis selectable (16750 compatible) Programmable Auto Flow Control Mode (Auto CTS and Auto RTS).
- Support for any serial data baud rate, subject to the serial clock frequency, as follows: baud rate = (serial clock frequency) / (16 * divisor)
- Extended diagnostic Loopback Mode allows testing more Modem Control and Auto Flow Control features.



The *DesignWare DW_apb_uart Databook* is available at:

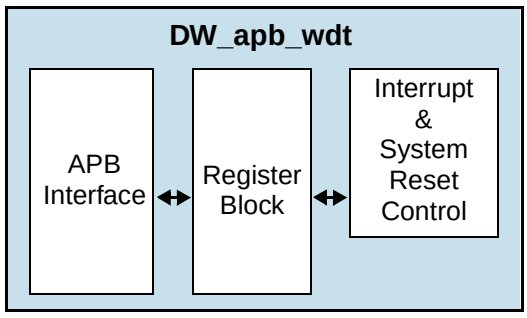
http://www.synopsys.com/products/designware/docs

AMBA Compliant

DW_apb_wdt

APB Watchdog Timer

- Configurable APB data bus widths of 8, 16, and 32 bits.
- Configurable watchdog counter width of 16 to 32 bits.
- Counter counts down from a pre-set value to zero to indicate the occurrence of a timeout.
- Programmable timeout range (period). The option of hard-coding this value during configuration is available to reduce the register requirements.
- After a timeout, the DW_apb_wdt can perform one of the following operations:
  - Generate a system reset
  - First generate an interrupt and if this is not cleared by the time a second timeout occurs, then generate a system reset
- Component version ID register
- Optional external clock enable signal to control the rate at which the counter counts.
- Optional dual programmable timeout period, used when the duration waited for the first kick is different than that required for subsequent kicks. The option of hard-coding these values is available.
- Programmable and hard-coded reset pulse length.
- Prevention of accidental restart of the DW_apb_wdt counter.
- Prevention of accidental disabling of the DW_apb_wdt.
- Optional support for Pause mode with the use of external pause enable signal.
- Test mode signal to decrease the time required during functional test.



The *DesignWare DW_apb_i2c Databook* is available at:

http://www.synopsys.com/products/designware/docs

Memory IP

The following Memory IP are briefly described in this section:

Component Name	Component Description	Component Type
DW_memctl	Memory Controller (page 250)	Synthesizable RTL
DW_rambist	DesignWare Memory BIST solution (page 251)	Synthesizable RTL

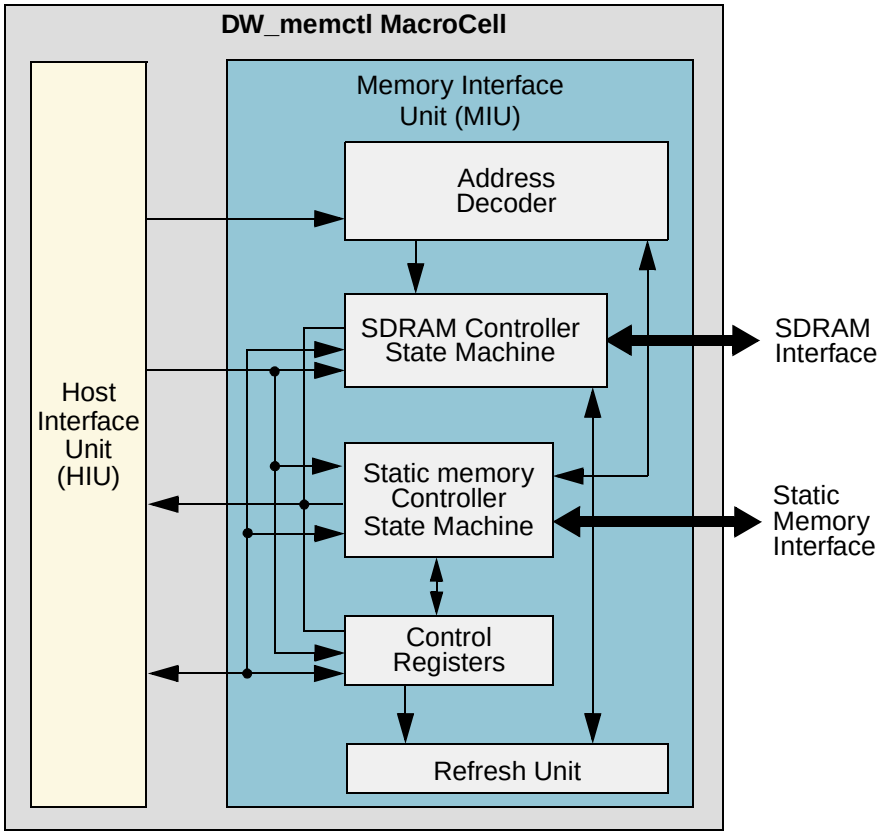
To view the complete DesignWare memory portfolio, refer to the following:

http://www.synopsys.com/products/designware/memorycentral

AMBA Compliant

DW_memctl
Memory Controller

- AMBA AHB bus-compatible
- AHB data widths of 32, 64, or 128 bits
- AHB address width of 32 bits
- Narrow access on wide AMBA bus
- Busy and early terminations
- Data width to 16, 32, 64, or 128
- SDR-SDRAM, MOBILE-SDRAM, SyncFlash, or Static memory interface
- Separate or shared address and buses between SDRAM and Static memories
- 1:1 or 1:2 ratios with AHB data width for SDRAM interface
- 1:1 to 1:16 ratios with AHB data width for Static memory interface



The *DesignWare DW_memctl MacroCell Databook* is available at:

http://www.synopsys.com/products/designware/docs

DW_rambist

Memory Built-In Self Test

Interfaces

- IEEE 1149.1 TAP controller interface
- Two clock interface, one for a slower TAP I/F, second for at-speed BIST execution
- Optional MUX block that supports either embedded multiplexers inside the memories or user-specified multiplexers
- Flexible configuration for embedded MUX block, providing a better interface to memory control signals with different widths and polarities

Error Diagnostics

- Pause on first and subsequent failures mode, serial debugging
- Failing address and data may be scanned out for examination
- Quick debug mode, continue on failures mode, failing addresses not recorded
- Parallel debug port to observe the failing memory data bits

BIST Tests

- User choice of March LR (14n), March C– (10n) and MATS++ (6n)
- Custom (user-defined) patterns option
- Optional SRAM retention test, (5n + delay), auto pause mechanism
- Selection of background and complement background data patterns
- Default sequence or run-time selection of individual test
- Improved test execution time through reduced memory read/write cycles (each access to synchronous memory occurs in one clock cycle)
- Configuration of Mode Register reset value to provide easy power-up tests
- Higher speed clock frequency

Supported Memories

- Synchronous and asynchronous SRAM
- Asymmetrical pipelining support, up to four stages
- Support for 32 memories per BIST controller
- Highly configurable memory interface to suit most types of memories

Supported Memory Configurations

- True at-speed testing of memories in parallel
- Memory array test via single port and multi-port
- Ability to enable/disable testing of individual memories
- Multiple controller scheduling
- Support for incomplete address space

DW_rambist
Memory Built-In Self Test

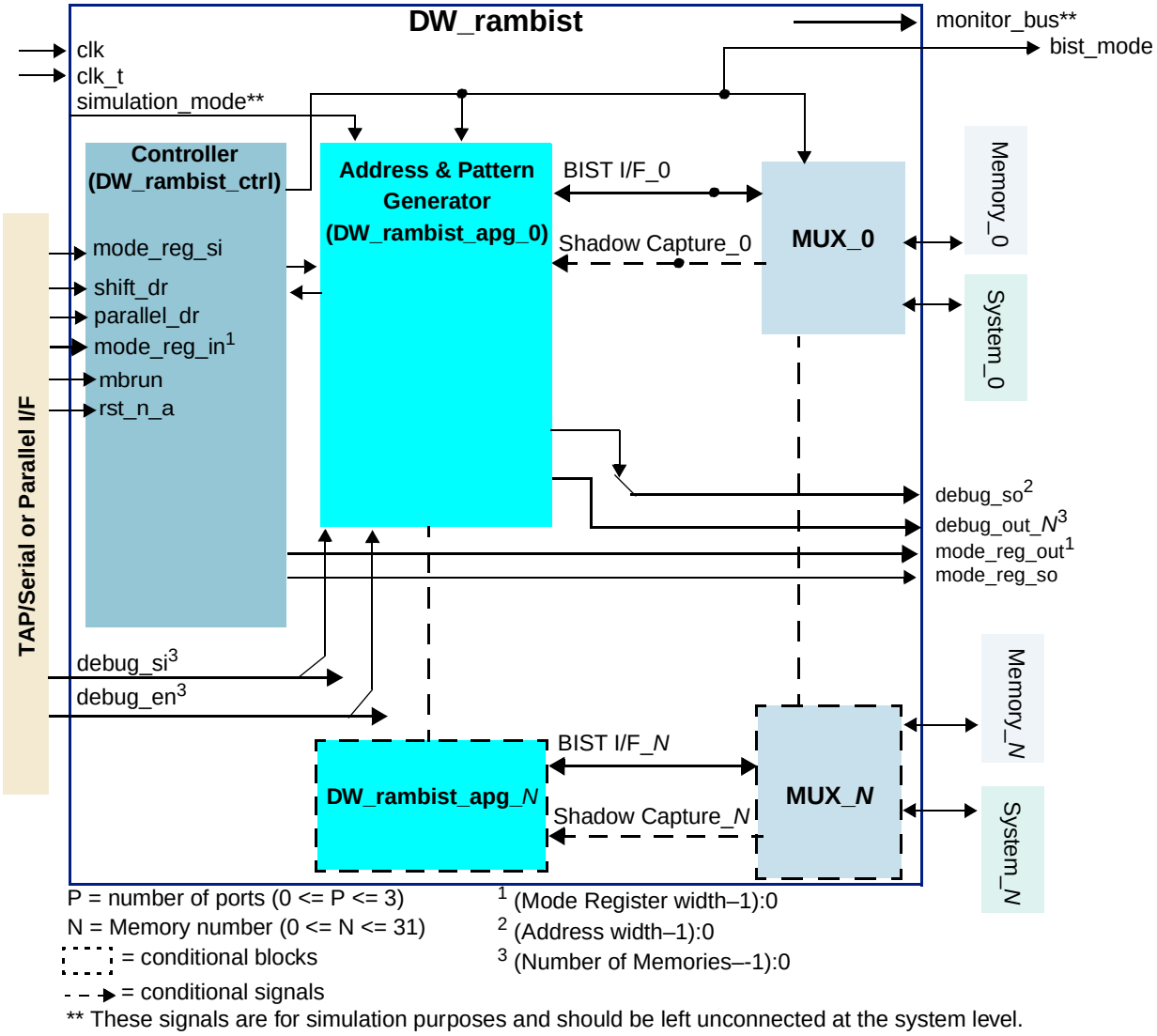
MacroCells

Design for test

- Configuration of shadow logic capture
- Sample script for scan chain creation and connection (part of example design)
- Integration with DFT Compiler, BSD Compiler, and TetraMax

Design for Verifiability

- simulation_mode signal to provide verification of very large configurations and to quickly check system-level interconnection



More information on the DW_rambist MacroCell can be found at:

http://www.synopsys.com/products/designware/docs/ds/i/DW_rambist_ds.pdf

Microprocessors/Microcontroller Cores

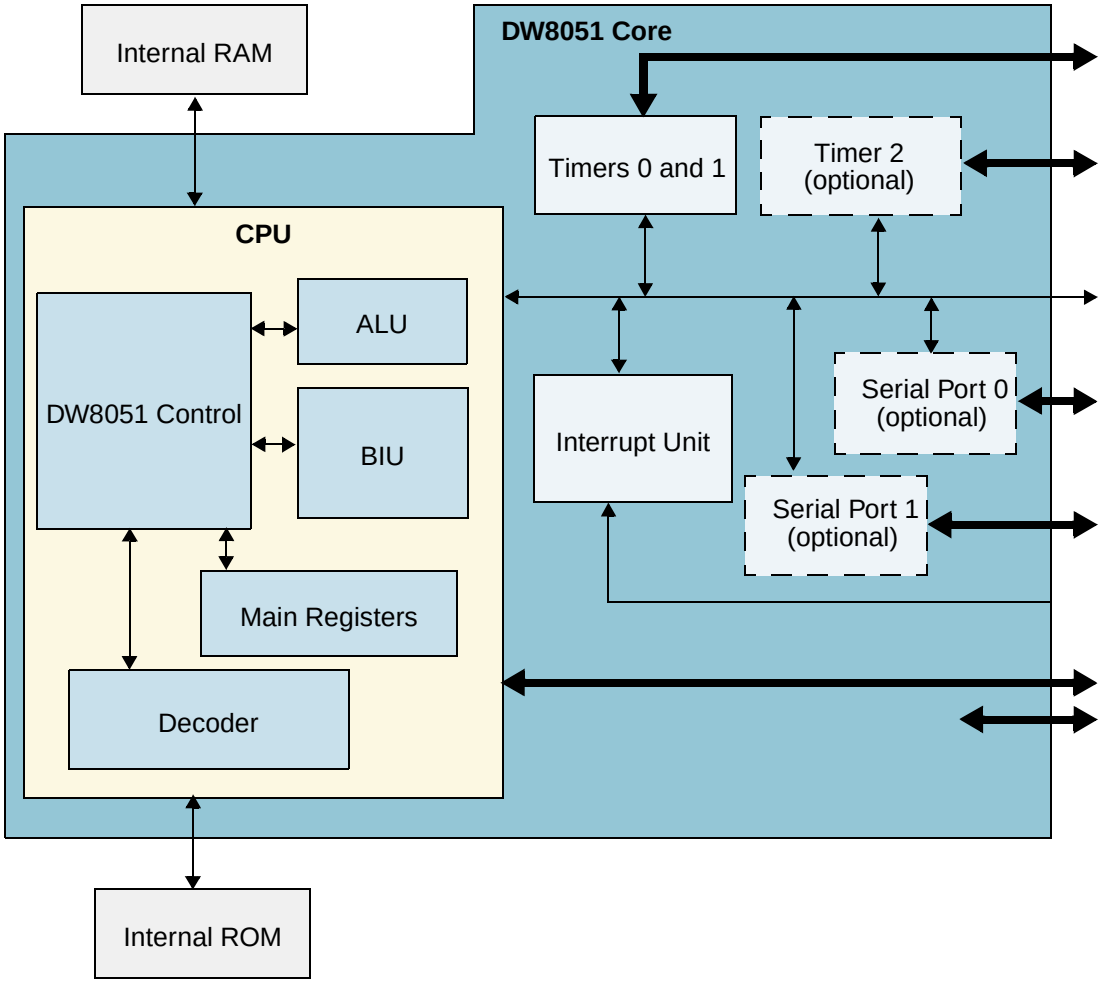
Synthesizable RTL of these cores are available through the Star IP Program. The components detailed in this section contain a page reference in the following table. For more information on this program, visit http://www.synopsys.com/designware.

Component Name	Component Description
DW_IBM440 (Available Q2'03)	PowerPC 440 Microprocessor Core from IBM (page 300)
DW_V850E-Star	V850E Processor Core from NEC (page 302)
DW_C166S	16-bit Processor from Infineon (page 304)
DW_TriCore1	TriCore1 32-Bit Processor Core from Infineon (page 306)
DW_MIPS4KE	Processor Core Family from MIPS (page 308)
DW8051	8-bit Microcontroller (See page 254)

DW8051

8051 Microcontroller

- Standard 8051 instruction set
- Configurable full-duplex serial ports
- High–speed architecture
- Parameterizable RAM address range
- Parameterizable ROM address range
- Fully static synchronous design
- Integration of user-defined peripherals through external Special Function Register (SFR) interface
- Enhanced memory interface with 16-bit address bus
- Variable length MOVX to access fast/ slow RAM peripherals



The *DesignWare DW_8051 MacroCell Databook* is available at:

http://www.synopsys.com/products/designware/docs

3 DesignWare Library Verification IP

Overview

The Designware Library Verification IP (Intellectual Property) is divided into the following categories:

- "DesignWare VMT Models" on page 256
- "DesignWare FlexModels" on page 262
- "DesignWare SmartModels" on page 268
- "DesignWare Memory Models" on page 267

DesignWare VMT Models

VMT (Verification Modeling Technology) models are bus functional models and monitors that can be instantiated in VERA or HDL testbenches. All VMT models have a common command interface style that allows you to easily integrate standard bus protocol devices into your system testbenches.

All VMT models support these features:

- Multiple command streams – Switch command control conditionally or unconditionally. Execute “Master” and “Slave” command streams. Stop command execution until new command streams are loaded.
- Verilog, VHDL, or VERA testbenches – VMT models run on these simulators: Synopsys VCS; MTI Verilog; Cadence NC-Verilog and Verilog-XL; Synopsys Scirocco; MTI VHDL; and Cadence NC-VHDL.
- Configurable message formatting – Enable or disable …
  - Message types–Errors, Warnings, Timing, X-handling, Notes, Protocol.
  - Message logs–Simulator transcript window and/or log files.
  - Message features–”Building blocks” of message content.
- Interrupt-driven testbenches – Waits for an event from the model and blocks commands until the event happens.

Some VMT models support additional features. Consult model feature lists in this quick reference or model datasheets for supported features:

- Multiple command channels – Simultaneously send and receive data (full duplex operation).
- Constrained random test – Configure testbenches to execute transactions, transaction sequences, or transaction choice sets, weighted by any configurable parameter. Provide file or random payloads for those transactions.

The following list identifies the VMT models that are part of the DesignWare AMBA On-Chip Bus (OCB) solution:

- AMBA Advanced High-Performance Bus (AHB) Models - ahb_master_vmt, ahb_slave_vmt, ahb_monitor_vmt, ahb_bus_vmt, ahb_act_monitor_vmt (See page 258)
- AMBA Advanced Peripheral Bus (APB) Models - apb_master_vmt, apb_slave_vmt, apb_monitor_vmt (See page 260)

The following list identifies additional VMT models:

- pciexpress_vmt
- sio_txrx_vmt, sio_monitor_vmt (See page 261)

A brief introduction to the AMBA On-Chip Bus can be found at the following location:

http://www.synopsys.com/products/designware/dw_amba.html

AMBA AHB Models

ahb_master_vmt, ahb_slave_vmt, ahb_monitor_vmt, ahb_bus_vmt, ahb_act_monitor_vmt

All Models

- Multiple command streams
- Verilog, VHDL, or VERA testbenches
- Configurable message formatting
- Interrupt-driven testbenches

AHB Master

- Data width: 8-1024 bits
- Single or burst transfers
- Burst rebuild capability
- Constrained random test transactions
- Compare with expected data

AHB Slave

- OK, Error, Retry, or Split responses
- Programmable wait states
- Configurable memory fill patterns
- FIFO memory at any memory location
- Constrained random test transactions

AHB Monitor

- Cycle-based or transaction-based event monitoring
- Protocol checking
- Incremental coverage reporting

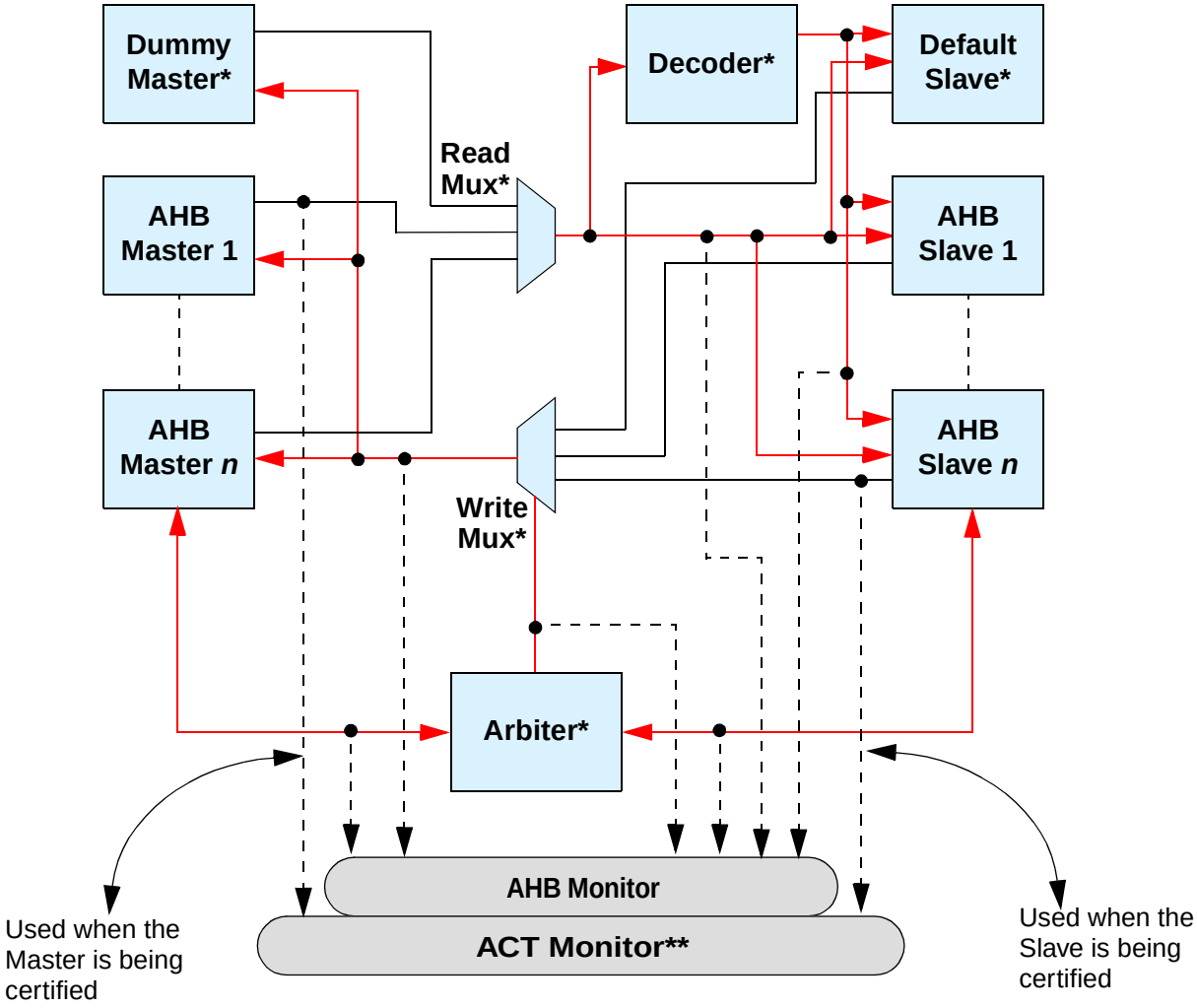
AHB Bus

- Up to 15 Masters and 15 Slaves
- Unlimited Slave memory maps
- Priority-based arbitration algorithm
- All types of Master transfers
- All types of Slave responses
- Configurable early burst termination and undefined length burst termination

AMBA Compliance Test-bench (ACT)

Note: This component is licensed separately from the DesignWare Library or DesignWare Verification Library.

- Certifies compliance with the AMBA 2.0 specification
- Generates compliance certificate as defined by ARM and Synopsys
- Multiple components may be certified during a single session
- Programmable coverage to reflect user configuration
- Re-certifies in customer environment without requiring an ACT license
- Coverage API enables self-checking from the test-bench
- Checks for protocol violations



* Dummy Master, Default Slave, Arbiter, Decoder, Write Mux, and Read Mux are part of the AHB Bus VIP model.

** This component is licensed separately from the DesignWare Library or DesignWare Verification Library.

The *DesignWare AHB Verification IP Databook* is available at:
http://www.synopsys.com/products/designware/docs

AMBA APB Models

apb_master_vmt, apb_slave_vmt, apb_monitor_vmt

All Models

- Multiple command streams
- Verilog, VHDL, or VERA testbenches
- Configurable message formatting
- Interrupt-driven testbenches

APB Master

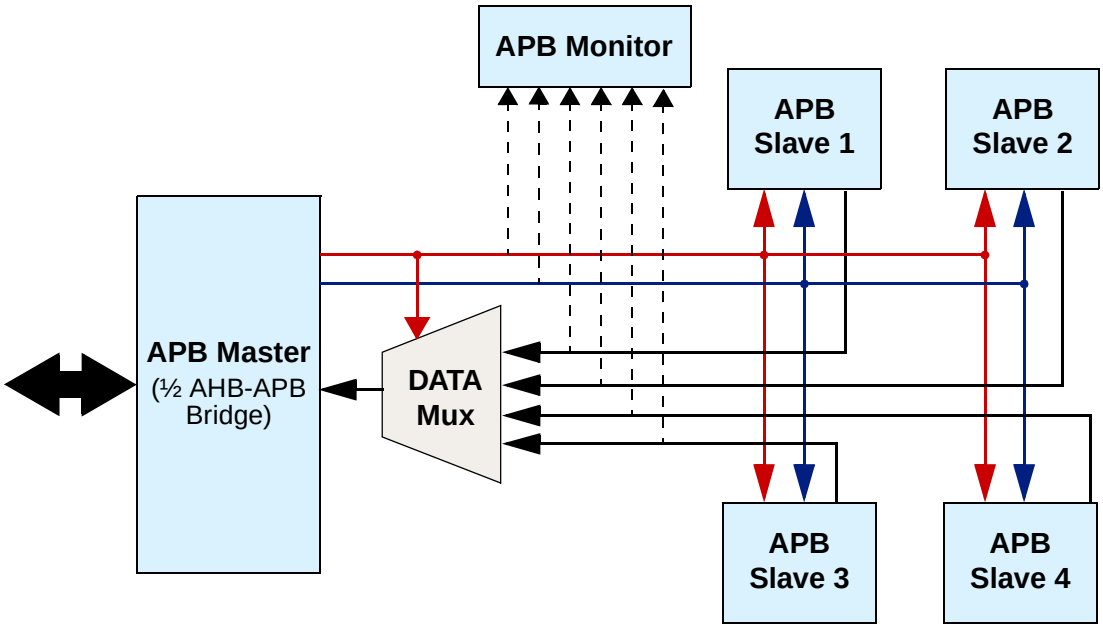
- 1-16 Slaves
- Data/Address width: 8-32 bits
- Constrained random test transactions
- Internal or external data mux
- Error injection capability

APB Slave

- Data/Address width: 8-32 bits
- Configurable memory fill patterns
- Big endian or little endian
- FIFO memory at any memory location

APB Monitor

- Transaction logging
- Protocol checking
- Incremental coverage reporting



The *DesignWare APB Verification IP Databook* is available at:
http://www.synopsys.com/products/designware/docs

Serial Input/Output Interface Models

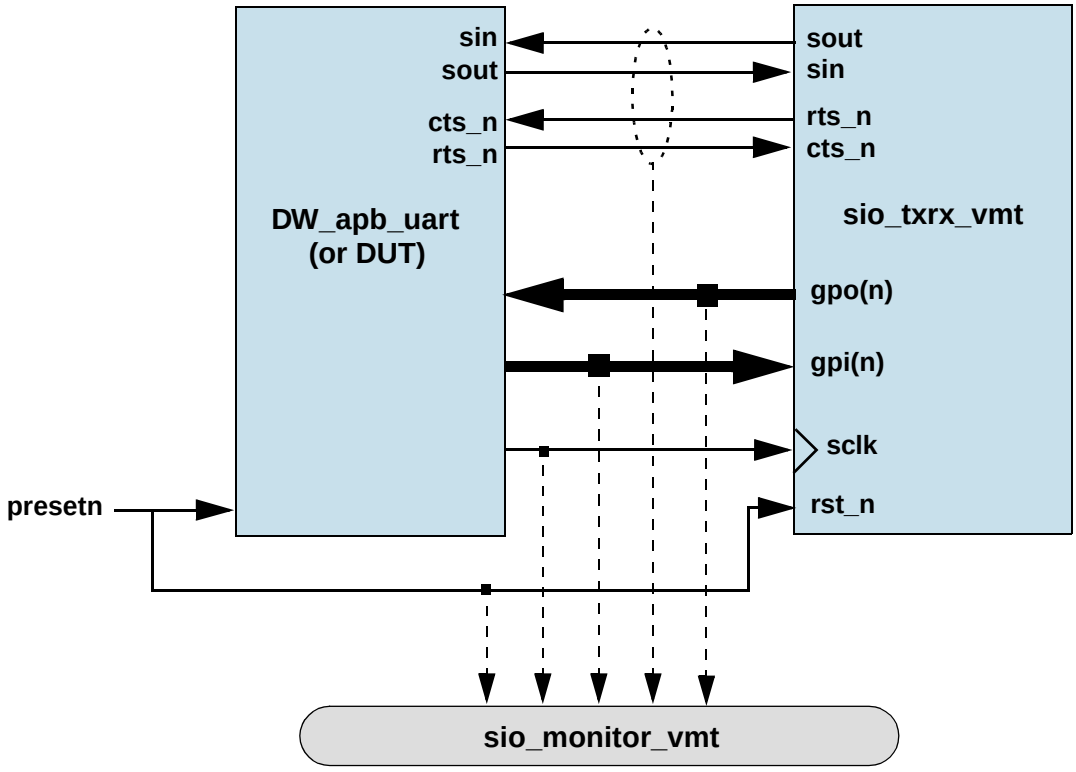
sio_txrx_vmt, sio_monitor_vmt

SIO TxRx Model

- Full duplex operation
- Fully configurable serial interface
- Both GPIO and SIO port interfaces
- Configurable receive FIFO depth
- Configurable internal baud clock
- Programmable hardware flow control
- IrDA SIR (infrared) mode support
- Error generation/injection capability
- Parity generate/check (odd/even/none)
- Robust command set control

SIO Monitor Model

- Protocol checking
- Transaction logging
- Watchpoint monitoring
- Configurable to match TxRx model
- Configurable internal baud clock
- Programmable hardware flow control
- IrDA SIR (infrared) mode support
- Parity generation and checking
- Command set control



The *DesignWare SIO Verification IP Databook* is available at:
http://www.synopsys.com/products/designware/docs

DesignWare FlexModels

FlexModels are binary simulation models that represent the bus functionality of microprocessors, cores, digital signal processors, and bus interfaces. FlexModels utilize the industry-standard SWIFT interface to communicate with simulators. FlexModels have the following features:

- Built with a cycle-accurate core and a controllable timing shell so that you can run the model in function-only mode for higher performance, or with timing mode enabled when you need to check delays. You can switch between timing modes dynamically during simulation using simple commands in your testbench.
- Feature multiple/different control mechanisms. You can coordinate model behavior with simulation events, synchronize different command processes, and control several FlexModels simultaneously using a single command stream.
- Allow you to use different command sources. You can send commands to FlexModels using processes in a Verilog or VHDL testbench, a C program, or a VERA testbench. You can switch between the HDL or VERA testbench and a compiled C program as the source for commands.

Listing of FlexModels

Table 1 lists the FlexModels that are available, including a brief description.

Table 1: Listing of FlexModels

Model Name	Vendor	Description
Processor Models		
arm7tdmi_fx	ARM	32-Bit RISC Microprocessor
mcf5307_fx	Motorola	MCF5307 Integrated Coldfire Microprocessor
mpc7400_fz	Motorola	MPC7400 RISC Microprocessor.
mpc740_fx	IBM	PowerPC RISC Processor
mpc750_l2_fx	IBM	MPC750 RISC Microprocessor L2 cache
mpc8260_fz	Motorola	MPC8260 Power Quic II Integrated PowerPC Processor
mpc860_fx	Motorola	32-Bit Quad Integrated Communications Controller
ppc603e_fx mpc603e_fx	IBM/ Motorola	PowerPC 603e RISC Microprocessor
tms320c6201_fx	TI	Digital Signal Processor

Table 1: Listing of FlexModels (Continued)

Model Name	Vendor	Description
vr5432_fx	NEC	MIPS RISC Microprocessor
vr5464_fx	NEC	64-bit Microprocessor
Bus Models		
enetrx_fx	Ethernet	Receives frame, stores received data and updates status.
enettx_fx	Ethernet	Handles data encapsulation and frame transmission.
enethub_fx	Ethernet	Emulates the protocol of Ethernet Hub at the pin and bus-cycle levels; handles data routing from TX to RX.
rmiis_fx	Ethernet	Interface between MII and reduced RMII interface.
ieee1394a_fx	IEEE1394a	Emulates the IEEE1394a protocol and bus nodes.
ieee1394phy_fx	IEEE1394a	Emulates multiple PHY layers connected to P1394a bus.
pcimaster_fx	PCI/PCI-X	Emulates the protocol of PCI/PCI-X initiators at the pin and bus-cycle levels. Initiates read and write cycles.
pcislave_fx	PCI/PCI-X	Responds to cycles initiated by the pcimaster_fx model or by the user's PCI master device.
pcimonitor_fx	PCI/PCI-X	Monitors, logs, and arbitrates activity on the PCI or PCI-X bus.
usbhost_fz	USB 1.1/2.0	Emulates the functions and timing of the Universal Serial Bus (USB) Host.
utmi_fz	USB 2.0/Intel	USB 2.0 Transceiver Macrocell Interface (UTMI) specifications from Intel. USB 2.0 interface between the Serial Interface Engine (SIE) and Macrocell.
Support Models		
sync8_fx	Synopsys	8-bit synchronization model

The individual DesignWare FlexModel databooks can be found with each model at:

http://www.synopsys.com/products/designware/ipdir

The *FlexModel User's Manual* is available at:

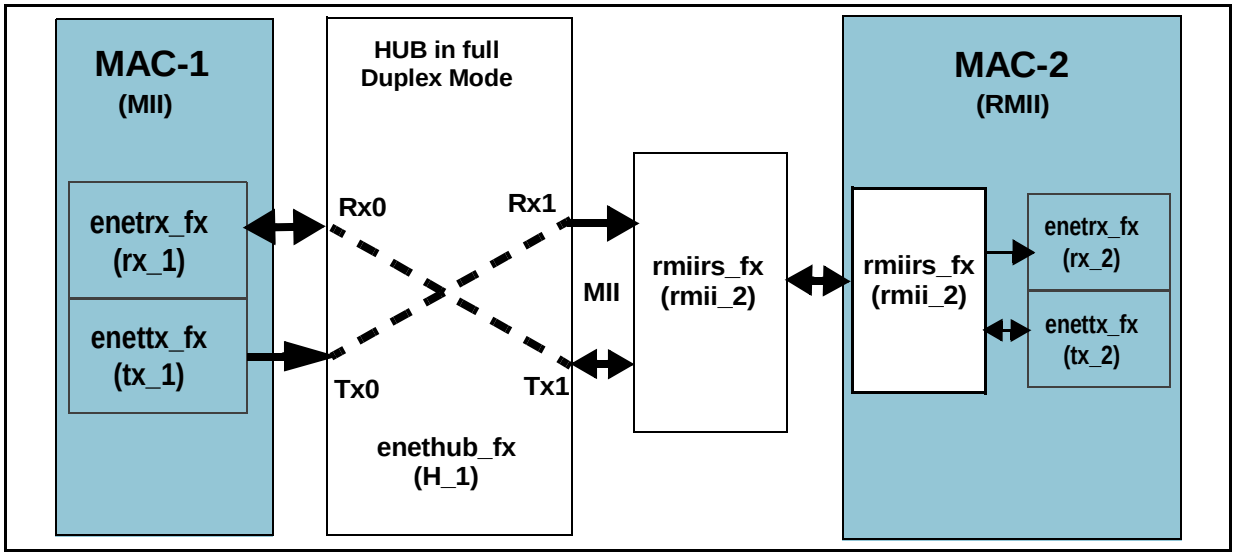
http://www.synopsys.com/products/designware/docs

Ethernet Verification Models

enettx_fx, enetrx_fx, enethub_fx, rmiirs_fx

The Synopys Ethernet Model set consists of four models and system testbenches in VERA, Verilog, C, and VHDL.

- enettx_fx. The enettx_fx FlexModel is the bus-functional model (BFM) for the Transmitter part of the IEEE 802.3 Ethernet MAC. This model is also compatible with the 802.3ab specification.
- enetrx_fx. The enetrx_fx FlexModel is the bus-functional model (BFM) for the Receiver part of the IEEE 802.3 Ethernet MAC. The following types of operations are performed by the enetrx_fx model during decapsulation: Address recognition, Frame check sequence validation, and Frame disassembly
- enethub_fx. The enethub_fx FlexModel provides the functionality of the 10/100 MBPS (MII). The following types of operations are performed by the model: acts as a common PHY for all MACs connected on its MII ports, and propagates the transmitted data from the transmitting MAC to all the MACs in the system.
- rmiirs_fx. The RMII interface is a low pin count MII interface intended for use between the ethernet PHY and switch (or repeater) ASICs. The interface has the following features: supports 10 Mb/s and 100 Mb/s data rates. single clock reference is sourced from the MAC to PHY (or from an external source); and, independent 2-bit wide transmit and receive paths.



The individual DesignWare FlexModel databooks can be found with each model at:

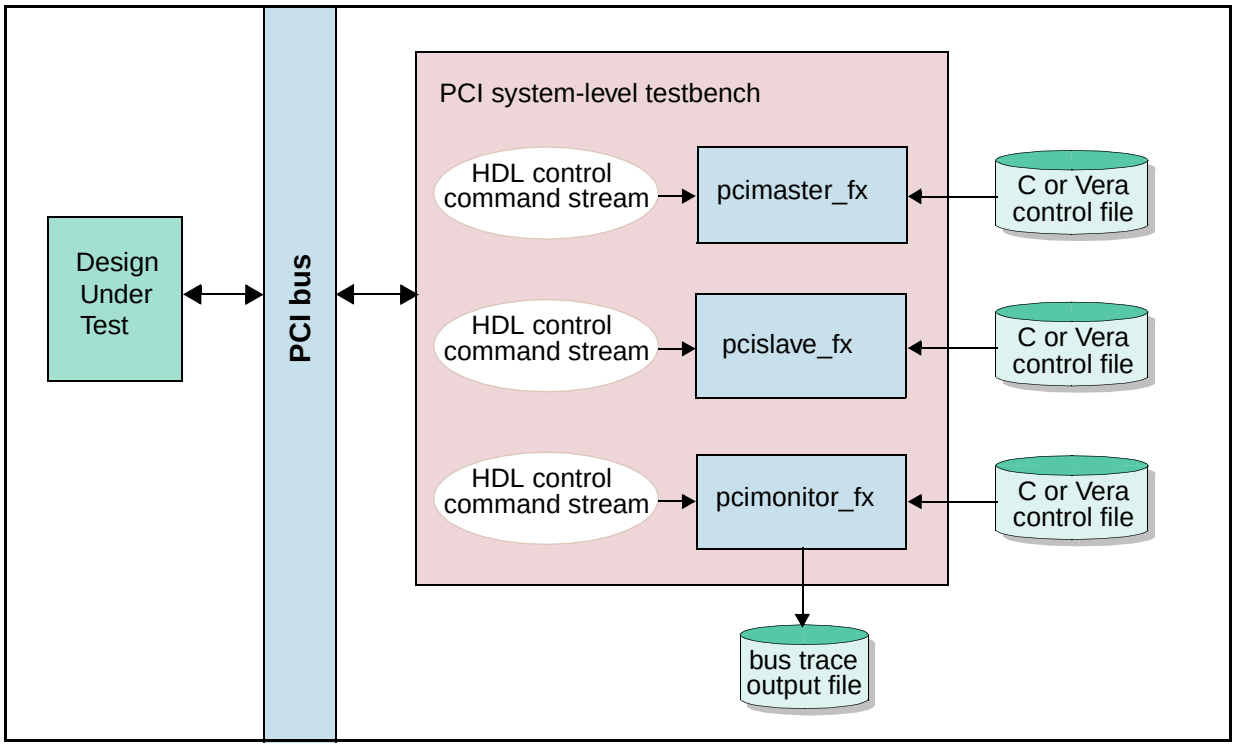
http://www.synopsys.com/products/designware/ipdir

PCI / PCI-X Bus Verification Models

pcimaster_fx, pcislave_fx and pcimonitor_fx

The Synopsys PCI/PCI-X FlexModel set consists of three separate PCI/PCI-X FlexModels and a set of system-level testbenches.

- **pcimaster_fx.** Performs timing violation checks and emulates the protocol of PCI/PCI-X initiators at the pin and bus-cycle levels. Initiates read and write cycles. In PCI-X mode, pcimaster_fx can function as a target for split transactions.
- **pcislave_fx.** Responds to cycles initiated by the pcimaster_fx model or by the user's PCI master device. In PCI-X mode, the pcislave_fx also functions as an initiator for split transactions.
- **pcimonitor_fx.** Monitors, logs, and arbitrates activity on the PCI or PCI-X bus.
- **PCI and PCI-X system testbenches.** Provides ready-to-use example testbenches for both conventional PCI mode and PCI-X mode. Each system testbench uses two pcimaster_fx models, two pcislave_fx models, and a pcimonitor_fx model.



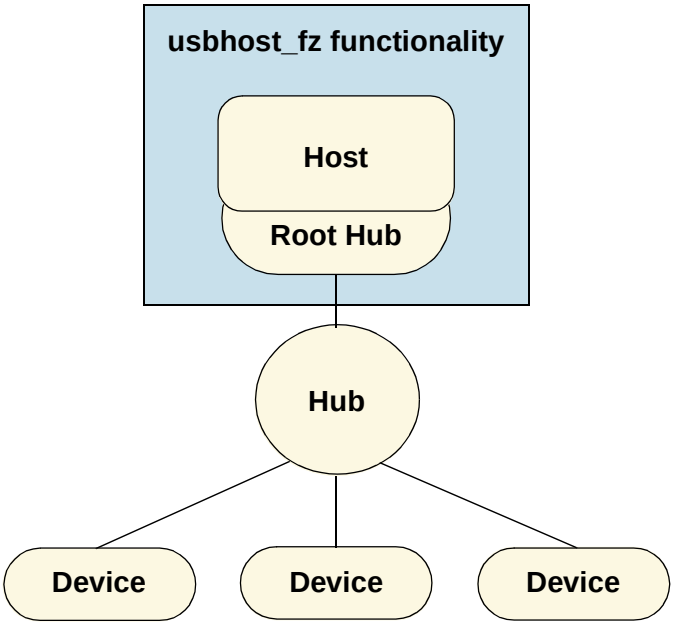
The individual DesignWare FlexModel databooks can be found with each model at:

http://www.synopsys.com/products/designware/ipdir

USB 1.1 / 2.0 Bus Host Interface Model

usbhost_fz

- Emulates the functions and timing of the Universal Serial Bus (USB) Host.
- Supports Revisions 1.1 and 2.0 of the USB Specification.
- Split, Control, Interrupt, Isochronous and Bulk transactions
- High-speed, full-speed, and low-speed transactions
- CRC generation on transmission
- CRC checking on reception
- Error generation (injection) for the following fields: Sync, PID, CRC5 in token packet, CRC16 in data packet, EOP, and Bit stuffing
- NRZI encoding with bit stuffing
- Automatic bit-rate change after preamble packet
- Programmable resume/suspend time
- One downstream port
- Three port states: awake (default), suspend, and resume
- Dynamic attach or detach (bus enumeration) using the usbhost_attach_detach command
- Internal data banks used by the model to perform bulk data transfers
- Status indication for IN and OUT



The individual DesignWare FlexModel databooks can be found with each model at:

http://www.synopsys.com/products/designware/ipdir

DesignWare Memory Models

DesignWare Memory Models are pre-verified simulation models of memory devices. The DesignWare Memory Models are built on top of the Synopsys MemPro memory model technology thus ensuring model accuracy, quality and reliability. There are thousands of pre-verified memory models to choose from, supporting over 25 memory vendors.

The models integrate with the simulator through the de facto industry standard SWIFT interface, which is supported by all Synopsys simulators and by all other major simulator vendors. Smarter verification is achieved by using the models debugging utilities.

You can search through the thousands of memory models using the memory model search capabilities offered as part of DesignWare Memory Central at: http://www.synopsys.com/memorycentral.

DesignWare Memory Models provide the following capabilities in your simulation environment:

- Dynamically managed 4-state (0, 1, X, or U) data storage. Models efficiently allocate workstation memory for all designs, whether low-density or memory-intensive.
- Word storage address ranges from 0 to 2^N-1, where N is an integer in the range 2 through 64. Supports integral word widths from 1 through 2048 bits.
- Unloading of a range of memory addresses. Unloaded memory addresses return to the uninitialized state and return the workstation memory associated with those addresses.
- Loading memory contents from or dumping memory contents to external files.
- Continuous control of model messages.
- Creation and deletion of traces on a range of addresses. When the simulation run performs a read or a write operation on a traced address, the model causes an event and provides the memory instance, address, data, and type of access.
- Means of interrogating and controlling the models through functions implemented as Verilog tasks, VHDL procedures, C functions, or VERA methods.

The Memory Model documentation is available at:

http://www.synopsys.com/products/designware/docs

DesignWare SmartModels

The SmartModel Library is a collection of over 3,000 binary behavioral models of standard integrated circuits supporting more than 12,000 different devices. The library features models of devices from the world's leading semiconductor manufacturers, including microprocessors, controllers, peripherals, FPGAs, CPLDs, memories, and general-purpose logic. SmartModels connect to logic simulators through the SWIFT interface, which is integrated with over 30 commercial simulators, including Synopsys VCS and Scirocco, Cadence Verilog-XL, and Mentor Graphics QuickSim II.

Instead of simulating devices at the gate level, SmartModels represent integrated circuits and system buses as "black boxes" that accept input stimulus and respond with appropriate output behavior. Such behavioral models are distributed in object code form because they provide improved performance over gate-level models, while at the same time protecting the proprietary designs created by semiconductor vendors.

All SmartModels and model datasheets are listed in the IP Directory, which you can find on the Web at:

http://www.synopsys.com/products/designware/ipdir/

SmartModel Features

- Support for "Windows" allowing you to view and change internal register values.
- Consistent SWIFT interface across most simulators.
- Simulation-efficient behavorial-level models.
- Industry-standard as well as configurable timing behavior.

SmartModel Types

There are two basic types of SmartModels:

- Full-functional Models (FFMs) simulate the complete range of device behavior.
- Bus-Functional Models (BFMs) simulate all device bus cycles. FlexModels are a type of BFM in the SmartModel Library, which you can control using Verilog, VHDL, VERA, or C.

For some devices, more than one type of model may be available, but these are exceptions, not the general rule. For detailed information about a specific SmartModel (including FlexModels), refer to the model's datasheet. For an overview of the FlexModels, see "DesignWare FlexModels" on page 262.

SmartModel Timing Definitions

All SmartModels have at least one timing version. To see what timing versions are available for a particular model, use the Browser tool to display a list of timing versions for that model.

If you need a timing version that is not supplied with the library, or if you want to back-annotate customized delays into the model's simulation, you can create a custom timing version as described in "User-Defined Timing" in the *Smartmodel Library User's Manual*.

Specific Model Information

SmartModel datasheets provide specific user information about each model in the library. The model datasheets supplement, but do not duplicate, the manufacturer's datasheets for the hardware parts. In general, the model datasheets describe:

- Supported hardware IP and devices
- Bibliographic sources used to develop the model (specific vendor databooks or datasheets)
- How to configure and operate the model
- Any timing parameters that differ from the vendor specifications
- How to program the device (if applicable) or otherwise use it in simulation
- Differences between the model and the corresponding hardware device

Models are partitioned by function, including:

- Processors/VLSI
- Programmables
- Memories
- Standards/Buses
- General Purpose

SmartModel datasheets have standard sections that apply to all models and model-specific sections whose contents depend on the model type.

4 DesignWare Cores

DesignWare Cores provide system designers with silicon-proven, digital and analog connectivity IP. Provided as heavily-annotated, synthesizable RTL source code, or in GDS format, these cores enable you to design innovative, cost-effective systems-on-chip and embedded systems. DesignWare Cores are licensed individually on a fee-per-project business model. The following table identifies the Synopsys Cores offering:

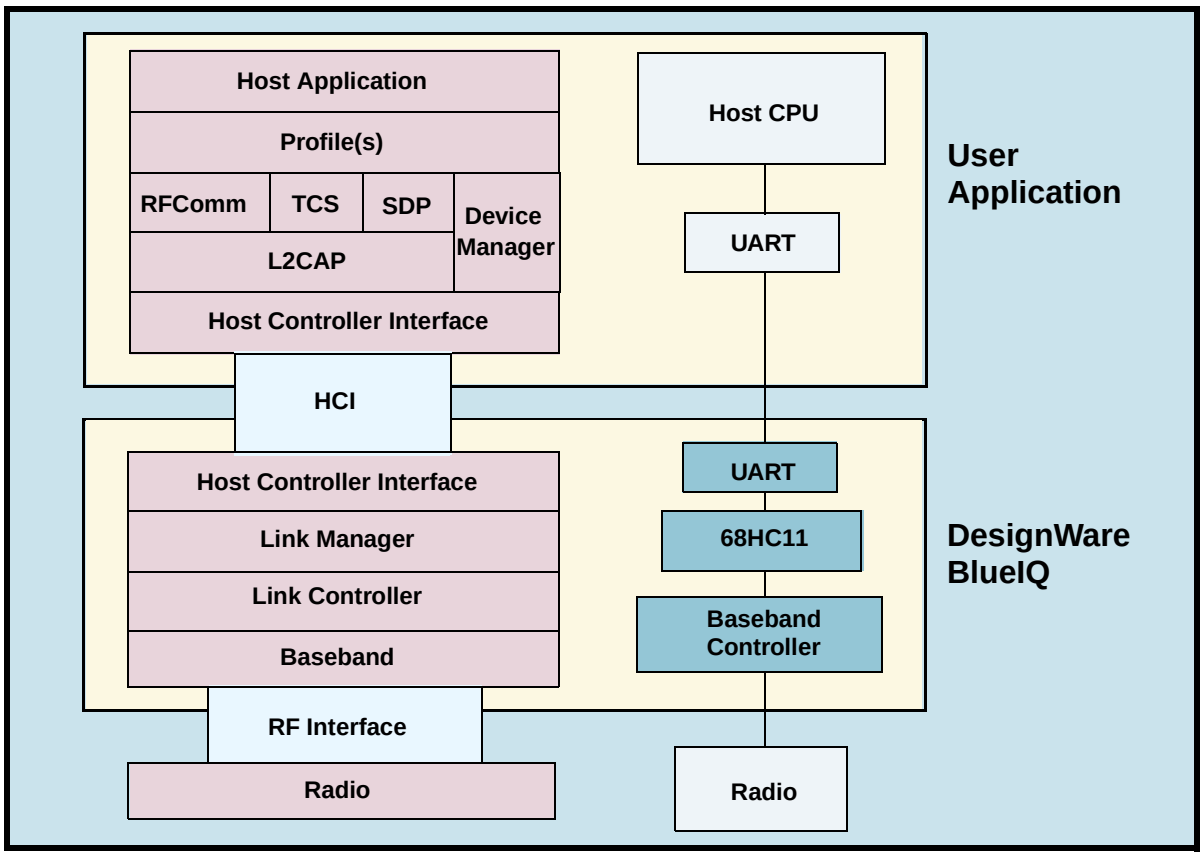
Component Name	Component Description	Component Type
Bluetooth Core		
dwcore_blueiq	Bluetooth Baseband and Link Manager (page 273)	Synthesizable RTL
dwcore_blueiq_devkit	Bluetooth Development Kit (page 275)	Hardware
Ethernet Cores		
dwcore_ethernet	Ethernet MAC, 10/100 Mbps Operation (page 276)	Synthesizable RTL
dwcore_ethernet_sub	Ethernet MAC Subsystem (page 277)	Synthesizable RTL
dwcore_gig_ethernet	Gigabit Ethernet MAC, 10/100-Mbps and 1-Gbps Operation (page 278)	Synthesizable RTL
dwcore_gig_ethernet_sub	Gigabit Ethernet MAC (GMAC) Subsystem (page 279)	Synthesizable RTL
PCI Cores		
dwcore_pci	32/64 bit 33/66 MHz PCI Core (page 280)	Synthesizable RTL
dwcore_pcix	32/64 bit 133 MHz PCIX Core (page 281)	Synthesizable RTL
dwcore_pci_express	Synthesizable PCI Express Core (page 282)	Synthesizable RTL

USB Cores		
dwcore_usb_otg	USB 2.0 Full Speed On-The-Go Controller Subsystem (page 283)	Synthesizable RTL
dwcore_usb1_device	USB 1.1. Device Controller (page 284)	Synthesizable RTL
dwcore_usb1_host	USB 1.1 OHCI Host Controller (page 285)	Synthesizable RTL
dwcore_usb1_hub	USB 1.1. Hub Controller (page 286)	Synthesizable RTL
dwcore_usb2_host	USB 2.0 Host Controller - UHOST2 (page 287)	Synthesizable RTL
dwcore_usb2_device	USB 2.0 Device Controller (page 288)	Synthesizable RTL
dwcore_usb2_phy	USB 2.0 PHY (page 289)	Hard IP
IEEE 1394 Cores		
dwcore_1394_avlink	IEEE 1394 AVLink (page 290)	Synthesizable RTL
dwcore_1394_device	IEEE 1394 Device Link Controller (page 291)	Synthesizable RTL
dwcore_1394_ohci	IEEE 1394 OHCI Link (page 292)	Synthesizable RTL
dwcore_1394_cphy	IEEE 1394 Cable PHY (page 293)	Synthesizable RTL
Java Acceleration Core		
dwcore_jvxtreme	Java™ Accelerator (page 294)	Synthesizable RTL
JPEG Cores		
dwcore_jpeg_codec	JPEG, CODEC (page 295)	Synthesizable RTL
dwcore_jpeg2_codec	JPEG2000 CODEC (page 296)	Synthesizable RTL
dwcore_jpeg2_encod	JPEG2000 Encoder (page 297)	Synthesizable RTL

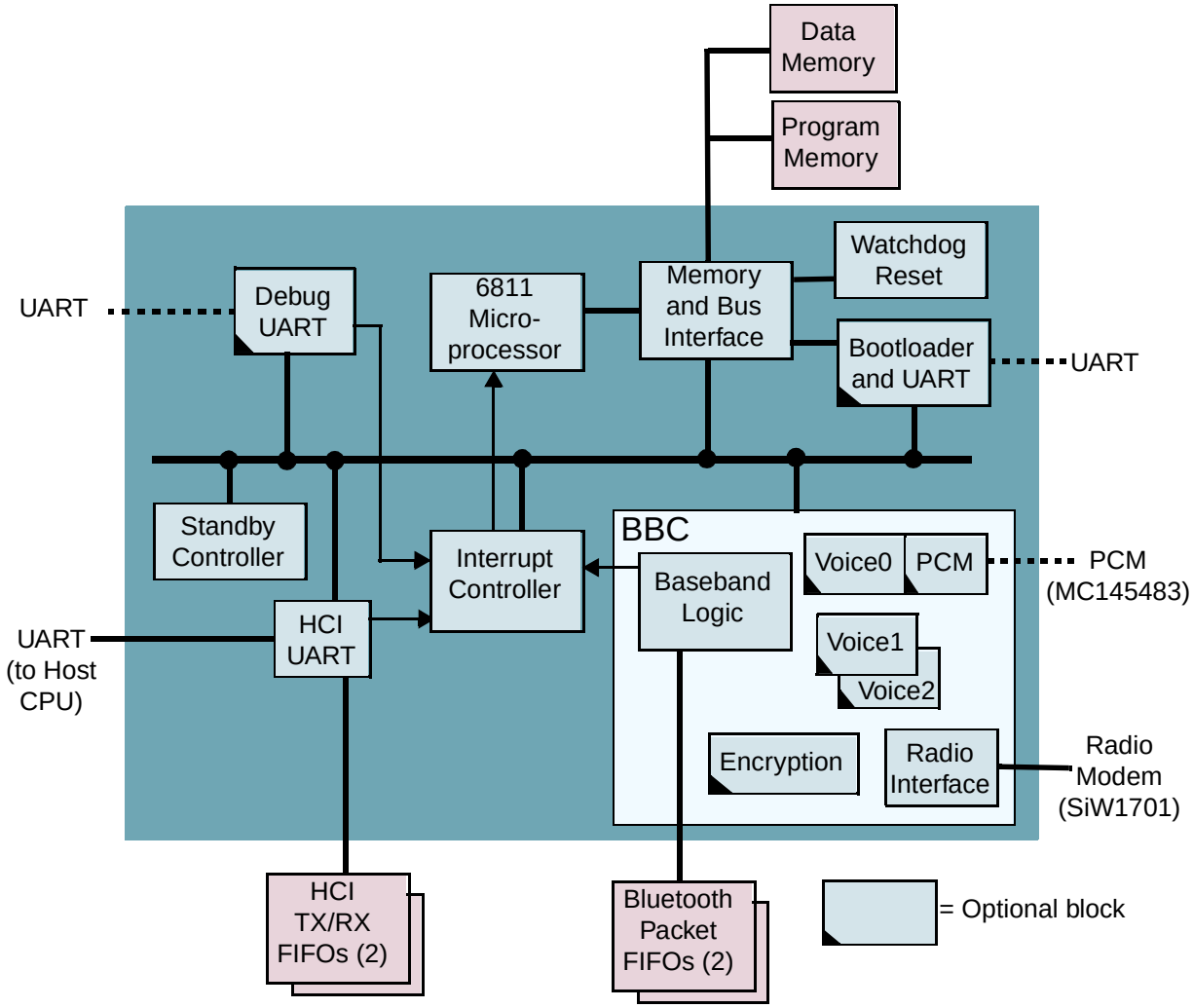
dwcore_blueiq

Synthesizable Bluetooth Baseband and Link Manager

- Provides two-CPU architecture for ease of integration
- Includes 6811-compatible microcontroller that offloads host processor of 100% of baseband and link manager processing
- Designed for low power and low system cost
- Simple HCI-over-UART interface that allows connection to any host
- Optimized connection to Silicon Wave SiW1701™ radio modem
- Small gate count and memory footprint
- Three voice channels and optional PCM, hardware encryption, and flexible memory interface for RAM, ROM, and Flash configurations
- Bluetooth development kit available on PC or other UART or USB-equipped hardware platform
- Qualified to Bluetooth v1.1 specification



Bluetooth System Block Diagram



BlueIQ Top Level Block Diagram

The dwcore_blueiq data sheet is available at:

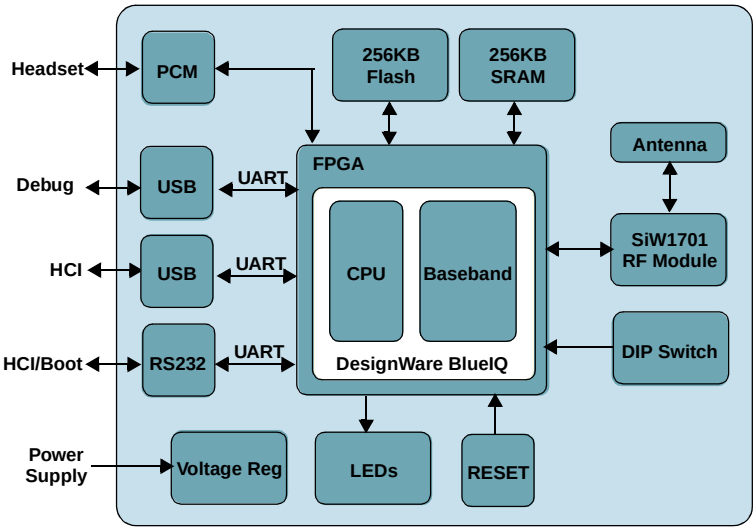
http://www.synopsys.com/products/designware/docs/ds/c/dwcore_blueiq.pdf

dwcore_blueiq_devkit

Bluetooth Development Kit

The Synopsys DesignWare Bluetooth Development Kit is a device used to develop applications for hardware platforms containing the DesignWare BlueIQ Core. It contains the DesignWare BlueIQ Core implemented in an FPGA and the Silicon Wave SiW1701™ Radio Modem. It can be connected to any PC or workstation over a standard USB or RS232 connection. Other features include the following:

- Simple serial connection to PC or other host development platform through UART or USB
- Headset port for audio connection
- Flash memory can be reprogrammed, allowing firmware updates to the BlueIQ Core
- FPGA can be reprogrammed, allowing hardware updates to the BlueIQ Core
- Debug connection provides access to menu of debugging commands
- Compatible with any Bluetooth upper stack software or development platform
- Includes Silicon Wave SiW1701 radio modem
- Includes an evaluation copy of the Mezoe Interface Express™ incorporating BlueStack® and Proto Developer™ software, for rapid application development



Bluetooth Development Kit

The dwcore_blueiq_devkit data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_blueiq_devkit.pdf

dwcore_ethernet

Synthesizable Ethernet Core

The Synopsys DesignWare Media Access Controller (MAC) includes the MAC and the MAC test environment. Features include the following:

- Collision detection in Half-Duplex mode (CSMA/CD protocol)
- Support for control frames in Full-Duplex mode (IEEE 802.3x)
- Preamble generation and removal
- Automatic 32-bit CRC generation and checking
- Configurable counters for remote monitoring (RMON)
- Complete status for transmission and reception packets
- Optimized for switching, routing, network interface card and system-on-chip applications
- Compliant with IEEE 802.3 and 802.3u specifications
- Supports 10/100 Mbps data transfer rates
- IEEE 802.3 Media Independent Interface (MII), Reduced Media Independent Interface (RMII) and General Purpose Serial Interface
- Supports Full- and Half-Duplex operations
- Virtual LAN (VLAN) support
- Power management: supports remote monitoring (RMON), LAN and magic packets
- RapidScript® utility for fast RMON customization
- Virtual Component Interface (VCI)
- Available in Verilog
- Application integration support
- Approximately 12K gates

The dwcore_ethernet data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_ethernet.pdf

dwcore_ethernet_sub

Synthesizable Ethernet Subsystem

The Synopsys DesignWare Ethernet Media Access Controller (MAC) Subsystem enables the host to communicate data using the Ethernet protocol (IEEE. 802.3). The subsystem is composed of three main layers: the DMA, the Transaction Layer Interface (TLI), and the Media Access Controller (MAC).

The Synopsy Ethernet MAC Subsystem enables Ethernet functionality for switch, NIC and system-on-chip applications. Ethernet MAC implements more than the traditional functionality of standard MACs: including a MAC Host, Station Management, Address Check, and Control/Status Register (CSR) blocks. These additional blocks provide the higher-level system functionality that is traditionally implemented in firmware or using separate products. With these additional capabilities, the Ethernet MAC simplifies the system implementation effort. Features include the following:

- Generic 32-bit single channel DMA engine
- Available in PVCI or AHB Interface
- Uses descriptor architecture for minimum CPU intervention
- Supports programmable interrupt options for different operational conditions
- Includes two dual-port FIFOs (one for transmission and one for reception)
- Optimized for switching, routing, network interface card, and system-on-chip applications
- Compliant with IEEE 802.3 and 802.3u specifications
- Supports 10/100 Mbps data transfer rates
- IEEE 802.3 Media Independent Interface (MII), RMII and Serial Interface
- Supports Full- and Half-duplex operations
- Virtual LAN (VLAN) support
- Power management: supports remote wake-up LAN and magic packets
- RapidScript® utility for fast customization
- Available in Verilog
- HomePNA (2.0) support with specific HPNA PHYs

The dwcore_ethernet_sub data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_ethernet_sub.pdf

dwcore_gig_ethernet

Synthesizable Gigabit Ethernet Core

- Collision detection and auto-retransmission on collisions in Half-Duplex mode (CSMA/CD) protocol
- Flow control using backpressure in Half-Duplex mode and flow control frames (IEEE 802.3x) in Full-Duplex mode
- Preamble generation and removal
- Automatic 32-bit CRC generation and checking
- Configurable counters for remote monitoring (RMON) and Simple Network Management Protocol (SNMP) (optional)
- Complete status for transmission and reception frames
- Compliant with IEEE 802.3, 802.3u, and 802.3z specifications
- Supports 10-/100-Mbps and 1-Gbps data transfers in Full-Duplex and Half-Duplex modes
- Supports rate selection (10/100/1000) post silicon
- Supports IEEE 802.3q
- Virtual LAN (VLAN) tagged frame detection
- IEEE 802.3z compliant GMII
- Supports CSMA/CD protocol in Half-Duplex mode
- Supports 1-Gbps frame bursting in Half-Duplex mode
- Supports IEEE 802.3 flow control for Full-Duplex operation
- Automatic Pause Frame generation in Full-Duplex mode
- Back pressure support in Half-Duplex mode
- Supports magic frame and wake on LAN frame detection
- Ethernet frame statistic support for Management Information Base (MIB)
- Automatic CRC and pad generation
- Options to insert CRC and pad for Transmit frames
- Options for automatic pad stripping for Receive frames
- Supports jumbo framesSupports internal loopback on the GMII/MII interface for debugging
- Supports a variety of flexible address filtering modes
- Separate 32-bit status returned for transmit and receive frames

The dwcore_gig_ethernet data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_gig_ethernet.pdf

dwcore_gig_ethernet_sub

Synthesizable Gigabit Ethernet Subsystem

The Synopsys DesignWare Gigabit Ethernet MAC (GMAC) Subsystem enables the host to communicate data using the Gigabit Ethernet protocol (IEEE 802.3). The GMAC Subsystem is composed of three main layers: the Gigabit Ethernet Media Access Controller (GMAC), the MAC Transaction Layer (MTL), and the MAC DMA Controller (MDC). Other features include the following:

- Implements the IEEE 802.3z and 802.3u specifications that define the 10-Mbps, 100-Mbps and 1-Gbps Ethernet standards
- Generates and accepts Control Frames in Full-Duplex Mode (IEEE 802.3x)
- Includes configurable counters for statistical network management support
- Provides complete status for transmission and reception packets
- Highly programmable DMA engine to meet optimal bus performance Programmable descriptor-interrupt architecture minimizes CPU overhead
- Supports programmable interrupt options for different operational conditions
- Includes two dual-port, RAM-based FIFOs (one for transmission and one for reception)
- Optimized for switching, routing, network interface card, and system-on-chip applications
- Compliant with IEEE 802.3z and 802.3u specifications
- Supports 10-/100-Mbps and 1-Gbps data transfer rates
- IEEE 802.3z Gigabit Media Independent Interface (GMII)
- IEEE 802.3z Physical Coding Sublayer (PCS) with Ten Bit Interface (TBI) that supports autonegotiation (optional)
- Supports Full- and Half-Duplex operations in all speed modes
- Supports Virtual LAN (VLAN) Detection
- Power management support: Remote Wake-up LAN and magic packets
- Synthesizable Verilog source code

The dwcore_gig_ethernet_sub data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_gig_ethernet_sub.pdf

dwcore_pci

Synthesizable Universal PCI Controller

The Synopsys DesignWare PCI intellectual property (IP) products are Verilog RTL synthesizable modules that provide an interface between the application and the PCI bus. Features include the following:

- PCI specification 2.3 compliant
- 15 application-optimized PCI IP, available in Verilog
- Silicon-proven 33-MHz and 66-MHz performance
- 32-bit or 64-bit PCI bus path
- 32-bit or 64-bit application data path
- Zero Latency, Fast Back-to-Back transfers
- Zero Wait-State Burst Mode transfers
- Support for Memory Read Line/ Multiple and Memory Write and Invalidate commands
- Dual Address cycles
- Loadable configuration space
- Universal configuration optimized for use in both Host Bridge and Add-in Card designs
- Delayed Read support
- PCI power management support
- PCI multifunction support

The dwcore_pci data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_pci.pdf

dwcore_pcix

Synthesizable PCI-X Controller and Test Environment

The Synopsys DesignWare PCI-X Controller is a set of Verilog RTL synthesizable building blocks ASIC designers use to implement a complete PCI-X interface. PCI-X is highly suitable in a wide range of applications, such as SCSI, Fibre Channel, Gigabit Ethernet, and graphics. Other features include the following:

- PCI-X 1.0a compliant
- Host Bridge functionality
- PCI 2.3 compliant
- 32-bit or 64-bit PCI-X bus path
- 64-bit application data path
- Supports 0-133 MHz PCI-X bus
- Supports up to 32 outstanding delayed/split transactions
- Dual Address Cycles (DAC)
- Message Signaled Interrupts (MSI)
- External EEPROM support
- Comprehensive Test Environment — Device Under Test linkable to the test environment
- RapidScript® parameterized configuration for fast customization
- Synthesizable Verilog source code

The dwcore_pcix data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_pcix.pdf

dwcore_pci_express

Synthesizable PCI Express Core

The DesignWare PCI Express Implementation IP Core is a synthesizable end-point solution that can be configured to address multiple applications, ranging from server and desktop systems to mobile devices. Other features include the following:

- Designed to PCI-Express specification for end-point solution
- Configurable lane width, x1-x4 @ 125 MHz, x1-x8 @ 250MHz
- Flexible buffer configuration
- 64 bit bi-directional application bus
- PIPE PHY connectivity
- Lane and polarity reversal in the physical and link layer
- Early detection of framing errors

More information is available at:

http://www.synopsys.com/products/designware/pciexpress.html

dwcore_usb_otg

Synthesizable USB 2.0 Full Speed On-The-Go Controller Subsystem

- The data buffer, configurable through RapidScript®, sizes FIFOs to specification, allowing further performance enhancement and gate count minimization.
- Flexible endpoint configuration allows designers to design one chip for each product, or single chips with alternate configurations that can be used in many products (such as a printer chip or a chip for use in printers, scanners, and multifunction printer/fax/scanners).
- The testbench proves the configured controller's USB and OTG compliance. Tests are designed for straightforward porting to an integrated design.
- Extensive USB compliance and interoperability testing in simulation
- Hardware state machines maximize Full Speed USB transaction throughput as compared to slower, software-based solutions
- Configurable data buffer and threshold options enable gate count fine tuning and performance maximization in one or many designs
- Flexible endpoint configuration allows one design for multiple applications, or for only one application
- Testbench is designed to enable porting of tests to a target environment for faster verification
- Verified in silicon

The dwcore_usb_otg data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb_otg.pdf

dwcore_usb1_device

Synthesizable USB 1.1 Device Controller

The Synopsys DesignWare USB Device Controller (UDC) is a set of synthesizable building blocks for implementing a complete USB device interface. Features include the following:

- 32-bit Virtual Component Interface (VCI)
- Maintains address pointer for endpoint 0 transactions
- Silicon proven
- USB 1.1 compliant
- AHB Interface & DMA Engine Options
- Standard register set Specification available
- Applications supported include: pointing devices, scanners, cameras, faxes, printers, speakers, monitor
- Available in Verilog
- Supports low-speed and full-speed devices
- Programmable number of endpoints
- Easily configurable endpoint organization
- Supports up to 15 configurations, up to 15 interfaces per configuration, and up to 15 alternate settings per interface
- Supports all USB standard commands
- Easy to add vendor/class commands
- Suspend/resume logic provided
- Approximately 12K gates for 5 physical endpoints
- Test Environment includes USB compliance tests and Bus Functional Models

The dwcore_usb1_device data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb1_device.pdf

dwcore_usb1_host

Synthesizable USB 1.1 OHCI Host Controller

The Synopsys DesignWare USB Host Controller (UHOSTI) Synthesizable IP is a set of synthesizable building blocks that ASIC/FPGA for implementing a complete USB OHCI Host Controller function. Features include the following:

- Silicon proven
- USB 1.1 Compliant
- VCI, AHB or Native interface
- Compatible with Open HCI 1.0 specification
- Available in Verilog
- Supports low-speed and full-speed devices
- Configurable root hub supporting up to 15 downstream ports
- Configuration data stored in Port Configurable Block
- Single 48-MHz input clock
- Simple application interface facilitates bridging the host to other system bus such as PCI, and the integration of the controller with chipsets and microcontrollers
- Integrated DPLL
- Support for SMI interruptspproximately 25K gates with 2 ports
- Test Environment includes USB compliance tests and Bus Functional Models

The dwcore_usb1_host data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb1_host.pdf

dwcore_usb1_hub

Synthesizable USB 1.1 Hub Controller

The Synopsys DesignWare USB Hub (UH01) is a set of synthesizable building blocks that ASIC/ FPGA for implementing a complete USB hub. The RapidScript® utility enables designers to easily configure the device by setting the number of downstream ports. The Synopsys UH01 product consists of the Hub Repeater and the Hub Controller. The Hub Repeater is responsible for connectivity setup and tear-down and supports exception handling such as bus fault detection/recovery and connect/disconnect detect. The Hub Controller provides the mechanism for host to hub communication. Hub specific status and control commands permit the host to configure a hub and to monitor and control its individual downstream ports. Other features include the following:

- Silicon proven
- USB 1.1 compliant
- Available in Verilog
- Supports low speed and full speed devices on downstream ports
- Integrated DPLL for clock and data recovery
- Downstream device connect / disconnect detection
- Supports suspend/ resume for power management
- Supports one interrupt endpoint in addition to endpoint 0
- Approximately 12K gates, for four ports
- Test Environment includes USB compliance tests and Bus Functional Models

The dwcore_usb1_hub data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb1_hub.pdf

dwcore_usb2_host

Synthesizable USB 2.0 Host Controller

The Synopsys DesignWare USB Host Controller (UHOST2) is a set of synthesizable building blocks that ASIC/FPGA for implementing a complete USB 2.0 host for 480-Mbps speeds. The UHOST2 can be customized and optimized as a stand-alone host chip or as an integrated ASIC for applications such as game consoles, set-top boxes, PCs, PDAs, and telecommunications equipment. In addition, the design can be easily processed in most technologies and can be easily bridged to any industry-standard bus and includes both the PCI and ARM® AHB interfaces. The application interface screens USB host controller design complexities, making it easy to integrate the UHOST2 device to customer target applications. Other features include the following:

- USB 2.0, EHCI, and OHCI specification compliant
- High-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) capability
- Configurable root hub supporting up to 15 downstream ports with 1.1 or 2.0 speed capability
- Choice of micro-frame or frame caching of data structures (EHCI)
- Simple application interface facilitates bridging the controller to other system buses
- PCI and AHB interfaces available
- Approximately 130K gates for a typical two-port implementation
- Compatible with Synopsys' High-Speed Certified USB 2.0 PHY
- Verilog source code
- Test Environment includes EHCI compliance test and BFMs

The dwcore_usb2_host data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb2_host.pdf

dwcore_usb2_device

Synthesizable USB 2.0 Device Controller

The USB 2.0 Device Controller (UDC20) features industry-standard interfaces that easily integrate the USB 2.0 transceiver and application logic. The RapidScript® utility builds the core and test environment in source code for the targeted application. Other features include the following:

- Certified High-Speed USB 2.0 Device Controller
- Supports 480 Mbps, 12 Mbps, and 1.5 Mbps devices
- Supports USB 2.0 Transceiver Macrocell Interface (UTMI)
- Available in Verilog
- Interfaces to any application bus
- Supports Virtual Component Interface (VCI) to application logic
- Optional support for AHB and DMA engine
- Programmable number of endpoints
- Flexible endpoint configuration with Windows® 98, ME, 2000, XP Host Class Drivers
- Process independent and portable
- Fully synchronous design
- Microprocessor and tool independent
- Backward USB 1.1 Specification compliant
- Supports up to 16 configurations, 16 interfaces per configuration, and 16 alternate settings per interface
- Easily configurable endpoint configuration
- Supports chirp sequences
- Supports Ping protocol
- Suspend/resume logic provided
- Supports UTMI-compliant and Philips ISP1501 Peripheral Transceivers
- Get Descriptor command can be decoded by the application
- Supports vendor-specific commands
- Maintains address pointer for endpoint 0 transaction

The dwcore_usb2_device data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb2_device.pdf

dwcore_usb2_phy

Synthesizable USB 2.0 Transceiver Macrocell Interface PHY

The USB 2 PHY includes all the required logical, geometric and physical design files to implement USB 2.0 capability in a System-on-Chip (SOC) design and fabricate the design in the designated foundry. The initial foundry process for the USB 2 PHY is the 0.18-micron CMOS digital logic process. Alternatively, design services are available for porting the USB 2 PHY to other semiconductor processes. The USB 2 PHY integrates high-speed, mixed-signal, custom CMOS circuitry compliant with the UTMI Specification (version 1.04), supports the USB 2.0 480-Mbps protocol and data rate, and is backward compatible to the USB 1.1 legacy protocol at 1.5 Mbps and 12 Mbps. Other features include the following:

- Complete mixed-signal physical layer (PHY) for single-chip USB 2.0 applications
- USB 2.0 Transceiver Macrocell Interface (UTMI) Specification compliant
- 8-bit interface at 60-MHz operation and 16-bit interface at 30-MHz operation chip
- Compatible with the Synopsys USB 2.0 Device and Host components
- USB 2.0 Device automatic switching between full- and high-speed modes
- Host Device automatic switching between full-, high- and low-speed modes
- Designed for minimal power dissipation for low-power and bus-powered devices
- Low-power design enables host enumeration of an unpowered device
- Sea-wall and decoupling structures reduce on-chip noise
- Suspend, Resume and Remote Wakeup mode support
- USB 2.0 test mode support
- Additional built-in analog testability features
- USB Implementers Forum certified

The dwcore_usb2_phy data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_usb2_phy.pdf

dwcore_1394_avlink

Synthesizable IEEE 1394 AVLink

The Synopsys DesignWare IEEE 1394 AVLink intellectual property (IP) is a set of highly configurable blocks that implements complete 1394 interface functions tailored to support audio/visual (AV)-oriented IEC 61883 applications. Configured through our RapidScript® utility, this device can also be optimized to act as a generic 1394 device controller. Therefore, AVLink can be effectively used in a wide range of applications, such as digital still cameras, video conferencing cameras, printers, scanners, digital audio devices, electronic musical instruments, digital VCRs/VTRs, and storage devices. Other features include the following:

- Silicon-proven IEEE 1394 Link Layer Controller for both audio/visual (A/V) and non-A/V applications
- Support for common isochronous packet (CIP) headers, time-stamping, and padded zeros for A/V data transactions
- IEEE 1394-1995 and 1394a-2000 specification compliance
- IEC 61883 requirement for A/V data streaming compliance
- Supports 100/200/400- Mbps data rates
- Full link layer implementation
- Asynchronous, isochronous, and PHY packet transmit and receive operations
- Cycle master and node controller capability
- Automatic isochronous resource manager detection
- Automatic acknowledge packet generation for received asynchronous packets
- Automatic 32-bit CRC generation and error detection interface
- Flexible, 32-bit Virtual Component Interface (VCI) for host; asynchronous and isochronous FIFO interface with burst and non-burst access modes
- Multi-speed, concatenated isochronous packet support
- Configurable number of isochronous transmit/receive channels
- Status reporting by extensive maskable interrupt register set
- Supports inbound and outbound single phase retry protocol
- RapidScript® custom IP configuration
- Verilog source code
- Optional 1394 simulation models

The dwcore_1394_avlink data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_1394_avlink.pdf

dwcore_1394_device

Synthesizable IEEE 1394 Device Link Controller

The Synopsys DesignWare IEEE 1394 Device Controller Link is a set of configurable blocks that ASIC/FPGA designers can use to implement a complete 1394 interface function for a target or host design. It can be effectively used in a wide range of applications such as digital still cameras, video conferencing camera, printers, scanners, digital audio devices, electronic music instruments, digital VCRs/VTRs, and storage devices. Other features include the following:

- Silicon-proven IEEE 1394 Device Controller
- IEEE 1394-1995 and 1394a-compliant
- Supports 100/200/400- Mbps 1394 bus rates
- Full implementation of Link layer
- Supports asynchronous, isochronous, and PHY transmit and receive packets
- Automatic 32-bit CRC generation and error detection
- Simple 8/16/32-bit application and isochronous interface with burst and non-burst access modes
- RapidScript® configuration for customization
- Synthesis scripts provided
- Cycle master capable
- Automatic IRM detection
- Hardware implementation of CSRs
- Power management features on application side
- Verilog source code
- Optional 1394 simulation models

The dwcore_1394_device data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_1394_device.pdf

dwcore_1394_ohci

Synthesizable IEEE 1394 OHCI Link

The Synopsys DesignWare IEEE 1394 OpenHCI (OHCI) Link enables PC host computers to communicate with 1394 peripherals via a standardized 1394a-2000 physical layer (PHY) and a common software driver set. The IEEE 1394 OHCI can be used in a standalone ASIC, or it can be integrated into an ASIC with a PHY targeted for high-volume, cost-effective 1394 applications. The IP technology is well suited for computer, multimedia, and mass storage applications requiring high bandwidth. Other features include the following:

- Complete IEEE 1394a- 2000 and 1394 OpenHCI standard rev. 1.0 support
- 100-/200-/400- Mbps bus speeds
- Eight isochronous transmit and eight isochronous receive DMA channels
- 128-byte, zero wait state burst operations
- 2-KB isochronous transmit FIFO
- 2-KB asynchronous transmit FIFO
- 4-KB receive FIFO
- VSIA-compliant Virtual Component Interface (VCI) for application backend
- OHCI driver support in Microsoft® Windows 98®, second edition
- Verilog source code

The dwcore_1394_ohci data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_1394_ohci.pdf

dwcore_1394_cphy

Synthesizable IEEE 1394 Cable PHY

The Synopsys industry-proven DesignWare 1394 Cable Physical Layer (CPHY) enables devices to interface with the 1394 serial bus. The 1394 CPHY is a synthesizable RTL design that provides all the necessary features to implement the complete IEEE 1394a specification for the digital portion of the cable PHY. The CPHY can be combined with an analog PHY and used in a standalone ASIC, or it can be integrated into an ASIC with a Link Layer controller. CPHY is well suited for multimedia and mass storage applications requiring high bandwidth, and is suitable for a wide range of applications, from basic low-cost devices (1 port) to sophisticated high-performance ASICs (up to 16 ports). Other features include the following:

- Complete IEEE 1394a support
- Supports 100/200/400-Mbps bus speeds
- Configurable number of ports (1 to 16)Simple, silicon-proven interface to mixed signal analog circuitry
- Supports suspend/resume protocol
- Supports Link-On LPS protocol
- RapidScript® configuration utility for design customization
- Synthesis scripts
- Verilog source code
- Approximately 14K gates (3 port)
- Proven in ASIC applications

The dwcore_1394_cphy data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_1394_cphy.pdf

dwcore_jvxtreme

Synthesizable Java™ Acceleration

The Synopsy next-generation DesignWare Java accelerator, JVXtreme®, is provided as synthesizable Verilog RTL for system integration. Features include the following:

- Used for set-top boxes, game consoles, and web phones/terminals, as well as mobile information devices such as PDAs, mobile phones, and smart cards
- Accelerates CPU performance for Java™ applications by up to 90x (peak) and 15x (average)
- Low-power consumption, with an average power of 200 µW/MHz in a 0.18 µm process
- Silicon-efficient design: 27K gates + memory
- Interfaces with any CPU and is compatible with any OS and Java Virtual Machine (JVM)
- Accelerates Java while preserving all previous CPU, operating system, and software application capabilities
- Better performance than a Just-in-Time (JIT) compiler-without JIT latency, code bloating, or memory requirements
- Synthesizable Verilog source code

The dwcore_jvxtreme data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_jvxtreme.pdf

dwcore_jpeg_codec

Synthesizable JPEG CODEC

The Synopsys DesignWare JPEG CODEC is part of an SoC-based multimedia solution that enables fast and simple image compression and decompression. The simplicity of the design allows for easy SoC integration, high-speed operation, and suitability for multimedia and color printing applications. Other features include the following:

- 100% baseline ISO/IEC 10918-1 JPEG-compliant
- Verified in hardware
- 8-bit channel pixel depths
- Up to four programmable quantization tables
- Single-clock Huffman coding and decoding
- Fully programmable Huffman tables (two AC and two DC)
- Fully programmable Minimum Coded Unit (MCU)
- Encoding/decode support (non-simultaneous)
- Single-clock per pixel encoding and decoding according to the JPEG baseline algorithm
- Support for up to four channels of component color
- Four-channel interface
- Simple external interface
- Low gate count-total gate count is 33K gates
- Stallable design
- Hardware support for restart marker insertion
- Support for single, grayscale components
- Internal register interface
- Fully synchronous design
- Available as fully functional and synthesizable VHDL or Verilog
- Includes testbench
- Individual Encoder and Decoder products are available from Synopsys

The dwcore_jpeg_codec data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_jpeg_codec.pdf

dwcore_jpeg2_codec

Synthesizable JPEG2000 CODEC

- Significantly crisper graphics and text than with JPEG, even at compression ratios exceeding 60:1.
- On-the-fly, lossless and lossy compression applications through the reversible 5/3 wavelet transform
- Fully compliant with the ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8) specification
- JPEG2000 encoding and decoding operations
- Reduced gate-count through real-time configuration of modules for encoding or decoding
- Programmable pixel depths for up to 10 bits per component
- Simple and efficient flow control for variable rate pixel and code input to the CODEC
- Configurable maximum tile and precinct dimensions
- Programmable code block size, precinct size, number of levels of decomposition, and quantization tables on individual tile basis to allow for customizable compression ratios
- Generates and parses inbitstream markers to minimize external header processing
- Packet header coding and decoding
- Arithmetic encoding termination at the end of each code block for higher compression ratios
- Easy to integrate CPU interface that allows access to all programmable registers
- Available in fully functional and synthesizable Verilog RTL
- Comprehensive test environment
- Fully synchronous design

The dwcore_jpeg2_codec data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_jpeg2_codec.pdf

dwcore_jpeg2_encod

Synthesizable JPEG2000 ENCODER

The Synopsys DesignWare JPEG2000 Encoder compresses input image tiles and generates packets of coded data, performing all computationally intensive tile encoding processes as specified in the JPEG committee standard ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8). The simplicity and configurability of the design facilitate easy System-on-Chip (SoC) integration and high compression quality, making it ideal for color printing applications such as facsimile and scanning, and digital image applications including Internet, photography, remote sensing, mobile/wireless, medical, and e-commerce markets. Other features include the following:

- Significantly crisper graphics and text than with JPEG, even at compression ratios exceeding 50:1
- Fully compliant with the ISO/IEC JTC 1/SC 29/WG1 (ITU-T SG8) standard
- Supports lossless and lossy compression applications through the reversible 5/3 wavelet transformer
- 10-bit pixel depths
- Simple and efficient flow control supports variable rates of pixel input to the encoder
- Configurable maximum tile and precinct dimensions
- Code block size, precinct size, number of decomposition levels, and quantization tables are programmable on an individual tile basis, enabling customizable compression ratios
- Generates in-bit stream markers to minimize external header processing
- Supports packet header information coding
- Arithmetic encoder termination at the end of each code block for higher compression ratios
- Easily integrated CPU interface allows access to all programmable registers
- Available as fully functional and synthesizable source code
- Includes a comprehensive test environment
- Fully synchronous design

The dwcore_jpeg2_encod data sheet is available at:

http://www.synopsys.com/products/designware/docs/ds/c/dwcore_jpeg2_encod.pdf

dwcore_jpeg2_encod
Synthesizable JPEG2000 ENCODER

5 DesignWare Star IP

Design engineers who use the DesignWare Library have the ability to evaluate and design easily at their desktop using the following high-performance, high-value IP cores from leading Star IP providers.

Component Name	Component Description	Component Type
DW_IBM440 (Available Q2'03)	PowerPC 440 Microprocessor Core from IBM (page 300)	Synthesizable RTL[a] Verification Model
DW_V850E-Star	V850E Processor Core from NEC (page 302)	Synthesizable RTL[a] Verification Model
DW_C166S	16-bit Processor from Infineon (page 304)	Synthesizable RTL[a] Verification Model
DW_TriCore1	TriCore1 32-Bit Processor Core from Infineon (page 306)	Synthesizable RTL[a] Verification Model
DW_MIPS4KE	Processor Core Family from MIPS (page 308)	Synthesizable RTL[a] Verification Model
Mezoe Interface Express	Interface Express™ Toolkit including BlueStack® Protocol stack with Profiles and ProtoDeveloper™ Software (page 310)	Software Source Code
SMSC USB 2.0 PHY	Standard Microsystems Corporation GT3100 USB 2.0 PHY Star IP MacroCell (page 311)	Hard IP

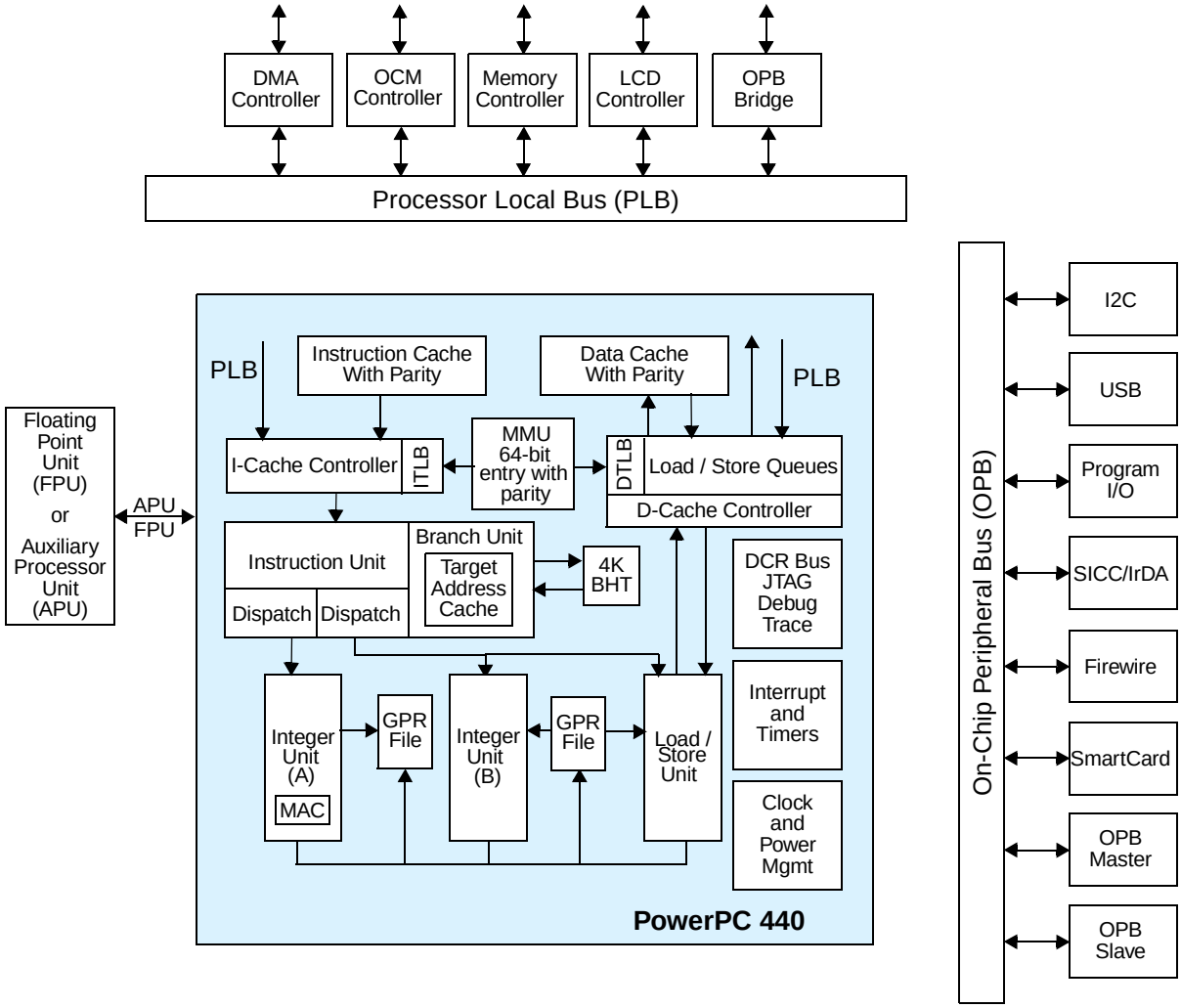
a. Verification models of these cores are included in the DesignWare Library. Synthesizable RTL of these cores are available through the Star IP Program.

DW_IBM440

DesignWare IBM PowerPC 440 Star IP Core

The PPC440x5 core is a high-performance, low-power engine that implements the flexible and powerful Book-E Enhanced PowerPC Architecture. Other features include the following:

- High performance, dual-issue, superscalar 32-bit RISC CPU
  - Superscalar implementation of the full 32-bit Book-E Enhanced PowerPC Architecture
  - Seven stage, highly-pipelined micro-architecture
  - Dual instruction fetch, decode, and out-of-order issue
  - Out-of-order dispatch, execution, and completion
  - High-accuracy dynamic branch prediction utilizing a Branch History Table (BHT)
  - Three independent pipelines
    - Combined complex integer, system, and branch pipeline
    - Simple integer pipeline
    - Load/store pipeline
  - Single cycle multiply
  - Single cycle multiply-accumulate (new DSP instruction set extensions)
  - Full support for both big and little endian byte order
  - Extensive power management designed into core for maximum performance/ power efficiency
- Separate 32-KB instruction and data caches
- Memory Management Unit with separate instruction and data micro-TLB’s
- Extensive hardware debug facilities incorporated into the IEEE 1149.1 JTAG port
- Timer facilities
  - 64-bit time base
  - Decrementer with auto-reload capability
  - Fixed interval timer (FIT)
  - Watchdog timer with critical interrupt and/or auto-reset
- Multiple core interfaces defined by IBM’s CoreConnect on-chip system architecture
  - Processor local bus (PLB) interfaces
  - Auxiliary Processor Unit (APU) Port
  - Device Control Register (DCR) interface for independent access to on-chip control registers
- JTAG , Debug, Reset and Trace interfaces
- Clock and power management (CPM) interface
- External interrupt controller (EIC) interface



DesignWare IBM PowerPC 440 Block Diagram

Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/ibm_powerpc.html

Star IP

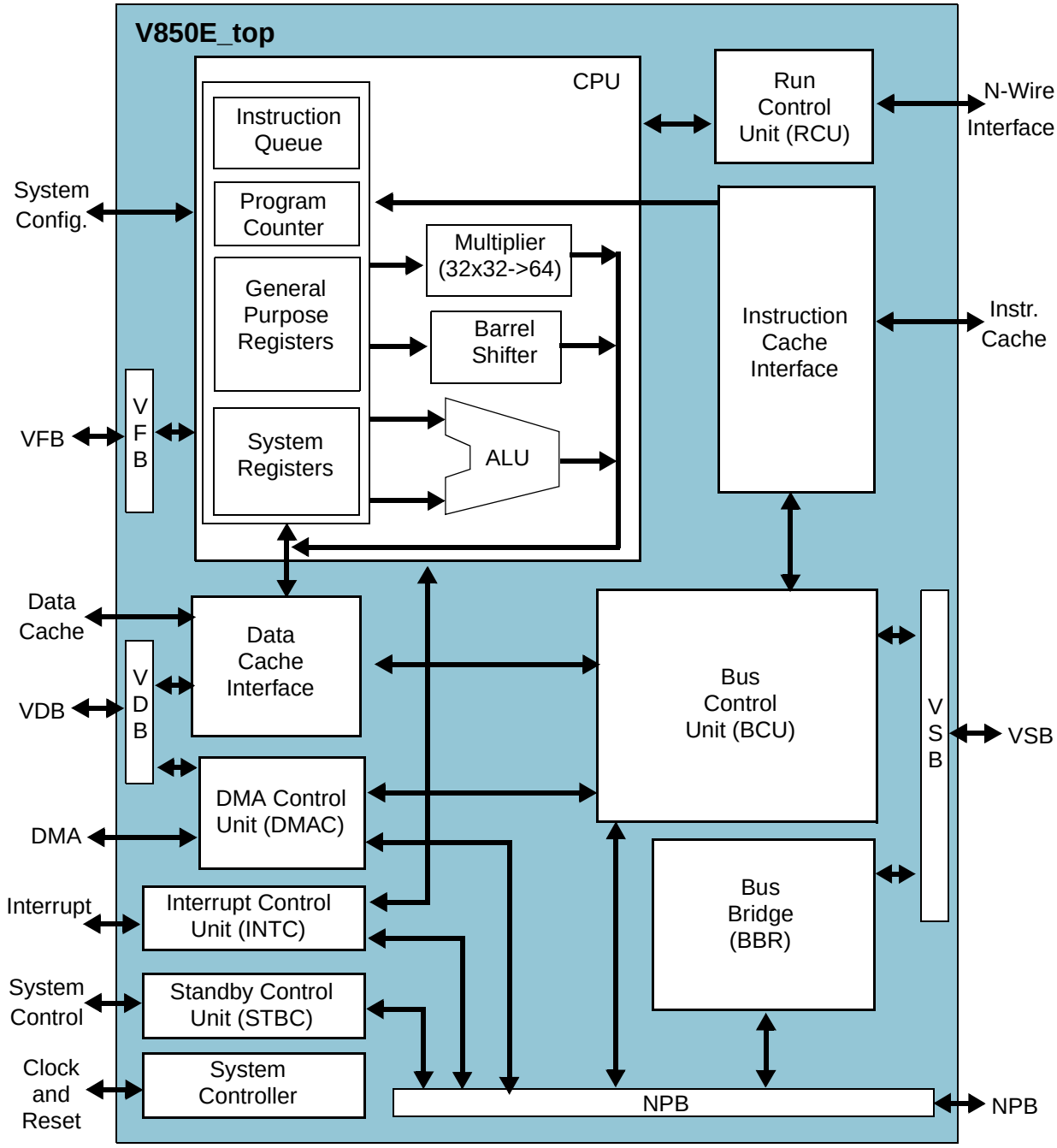
DW_V850E-Star

V850E Processor Core from NEC

The NEC V850E is a highly configurable, fully synthesizable RISC architecture microcontroller. The V850E core is ideal for applications that require high performance, low cost and minimum power consumption. In addition, the excellent processing horsepower of this 32-bit processor is perfect for new applications that need more than a 16-bit processor can provide. Other features include the following:

- Fully compatible with V850E1 instruction set
- Supports 32-bit and 16-bit instruction formats
- 64 MB, linear address program memory space
- 4 GB, linear address data memory space
- V850E1 CPU:
  - Five-stage pipeline
  - 32-bit datapath
  - Simultaneous transfer of instruction and data on separate buses (Harvard architecture)
  - RISC architecture plus special instructions for saturation, bit manipulation, and multiply (using integrated hardware multiplier)
  - 32 general-purpose, 32-bit registers
- Instruction cache (optional)
- Data cache (optional)
- Integrated 4-channel DMA (optional)
- Integrated interrupt controller supporting 3 external non-maskable interrupts (NMIs) and up to 64 external maskable interrupts
- Integrated run control unit (optional)
- Separate interfaces to internal ROM and RAM
- V850E System Bus (VSB) interface to high-speed peripherals
- NEC Peripheral Bus (NPB) interface to low-speed peripherals
- Power management through HALT instruction and hardware or software generated stop mode (implemented by standby control unit (STBC))
- Fully synchronous design

See the block diagram on the following page.



V850E-Star Block Diagram

Also see the following web page for additional information:

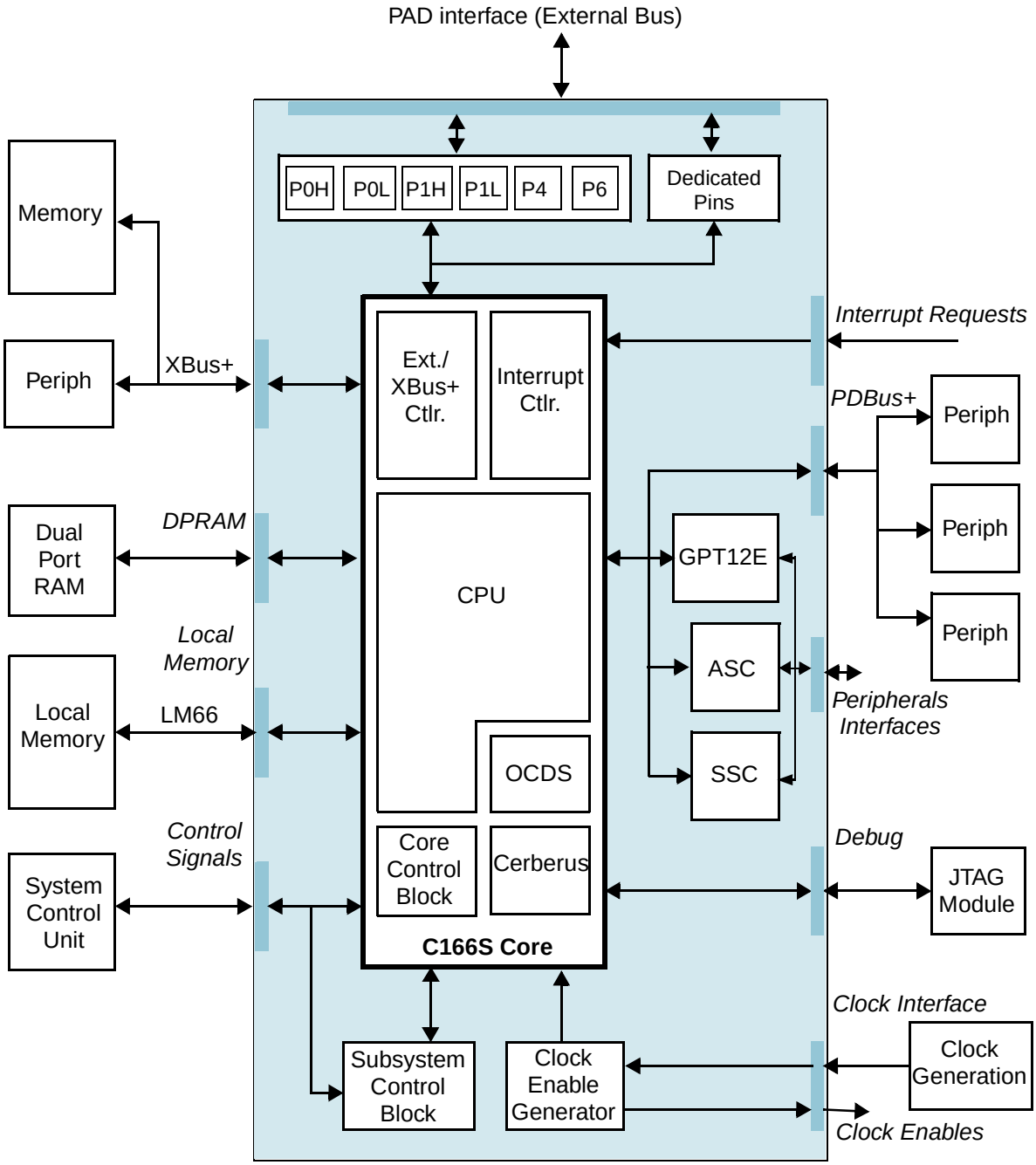
http://www.synopsys.com/products/designware/starip/nec_v850e.html

DW_C166S

16-Bit Processor from Infineon

The Infineon C166S is a highly configurable, fully synthesizable processor core based on the successful C166 microcontroller IC family and is 100% instruction-set compatible. Other features include the following:

- Four-stage pipelined, fully static, 16-bit CPU
- CPU speed up to 80 MHz (in 0.18-micron technology)
- Compatible with standard C166 instruction set
- Up to 16 MB addressable memory space
- Optional multiplier/accumulator (MAC) unit
- Integrated On-Chip Debugging System (OCDS)
- Most instructions execute in a single instruction cycle (2 CPU clock cycles)
- Multiple register banks with single-instruction-cycle context switching
- 16x16 multiplication in 5 instruction cycles, 32/16 division in 10 instruction cycles
- Multiple high-bandwidth internal data buses also available externally: XBus+, Local Memory bus, Dual Port RAM bus, PDBus+
- Up to 112 interrupt nodes (15 of which are used for internal interrupts)
- 16 level/8 group level interrupt priority
- Up to 16 interrupt-driven peripheral event controller (PEC) channels
- Power reduction modes
- Programmable watchdog timer
- Debug interface that supports hardware, software, and external breakpoints, and provides access to internal registers and memory through a JTAG module
- Support for a wide variety of third-party development and debugging tools (the current list of support tools is available at http://www.infineon.com)
- Asynchronous/synchronous serial channel (ASC)
- High-speed synchronous serial channel (SSC)
- General purpose timer block (GPT12E)
- Ports I/O module that provides programmable external bus or general purpose I/O port functionality



C166S Subsystem Block Diagram

Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/infineon_c166s.html

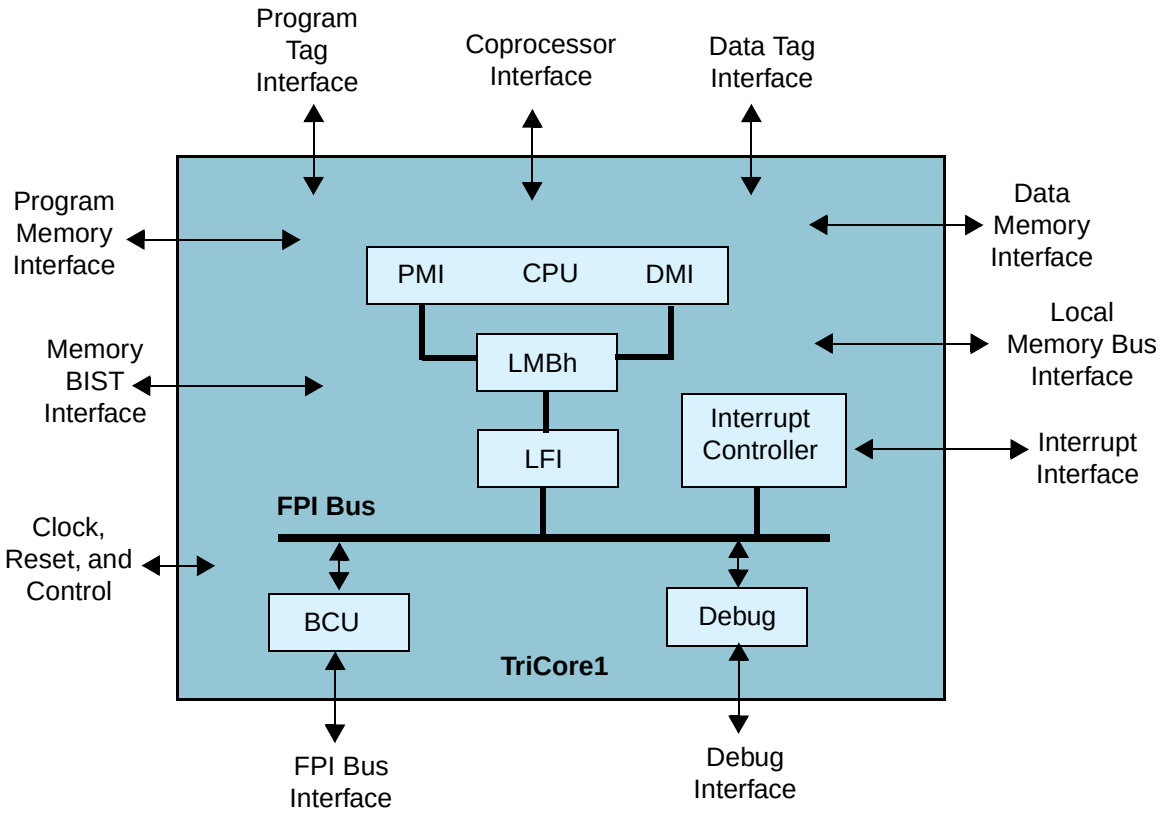
Star IP

DW_TriCore1

TriCore1 32-Bit Processor Core from Infineon

The Infineon TriCore1 is the first unified MCU-DSP architecture in a single core, ideally suited to SoC applications that require both microcontroller and DSP functionality. With high performance, low cost and minimal power consumption, the 32 bit TriCore1 superscalar processor meets the needs of automotive, industrial, mass storage, communications and other applications that require DSP and more processing power than a 16-bit processor can provide. Other features include the following:

- 2-bit load / store architecture
- Dual instruction use
- 4-GB address range
- General Purpose Register Set (GPRS)
  - Sixteen 32-bit data registers (Dx)
  - Sixteen 32-bit address registers (Ax)
  - Three 32-bit status & program counter registers (PSW, PC, PCXI)
- Shadow registers for fast context switching
- Automatic context save-on-entry and restore-on-exit for subroutine, interrupt & trap
- Two memory protection register sets
- Instruction formats: 16-bit and 32-bit
- Byte & bit addressing
- Saturation integer arithmetic
- Packed data
- Data and instruction caches (optional)
- Data types: boolean, integer with saturation, bit array, signed fraction, character, double word, signed, unsigned integers, IEEE-754 single precision floatingpoint
- Data formats: bit, byte (8 bits), half-word (16 bits), word (32 bits), double-word (64 bits)
- Zero overhead loop
- Instruction types: arithmetic, address, comparison, address comparison, logical, MAC, shift, coprocessor, bit logical, branch, bit field, load/store, packed data, system, MMU specific instructions
- Addressing modes: absolute, circular, bit reverse, long & short, base+offset, base+offset with pre & post-update
- Multiply & Accumulate (MAC) instructions: dual 16 x 16, 16 x 32, 32 x 32
- On-Chip Debug Support (OCDS) Levels 1 & 2



Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/infineon_tricore1.html

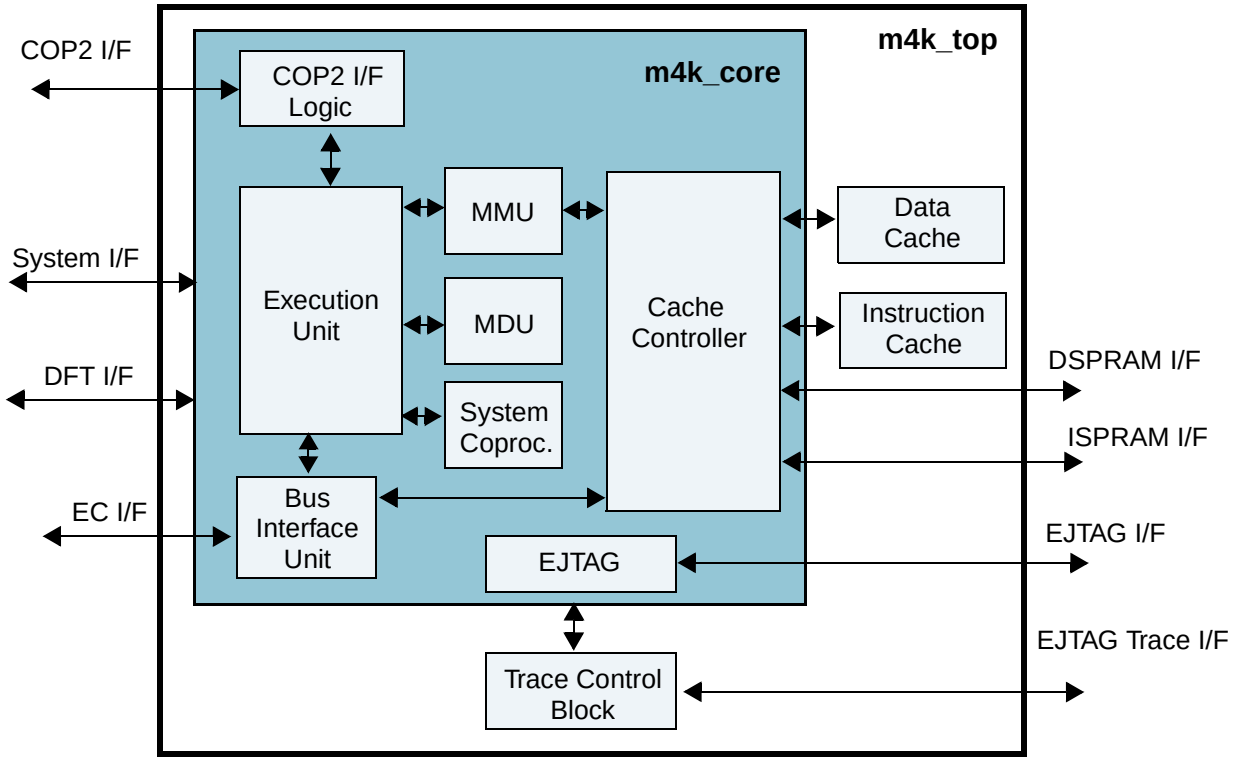
Star IP

DW_MIPS4KE

Processor Core Family from MIPS

The highly configurable MIPS32™ 4KE™ family represents the next-generation of 32-bit MIPS™ cores. Features include the following:

- 32-bit address and data paths
- Five-stage pipelined CPU
- Compatible with standard MIPS32 instruction set with optional support for MIPS16 instructions
- User-defined instructions (optional)
- Configurable instruction and data cache sizes
- MIPS R4000-style Privileged Resource Architecture
- Synchronous system (EC bus) interface
- Memory management unit:
  - Translation lookaside buffer (TLB) in 4KEc configuration
  - Fixed address mapping in 4KEm and 4KEp configurations
- Scratchpad RAM support (optional)
- Coprocessor 2 interface (optional)
- Multiply/divide unit:
  - High-performance implementation in 4KEc and 4KEm configurations
  - Area-efficient implementation in 4KEp configuration
- Power management
- Enhanced JTAG (EJTAG) debug support
- Support for a variety of third-party development and debugging tools (the current list of support tools is available at http://www.mips.com)



DW_MIPS4KE Block Diagram

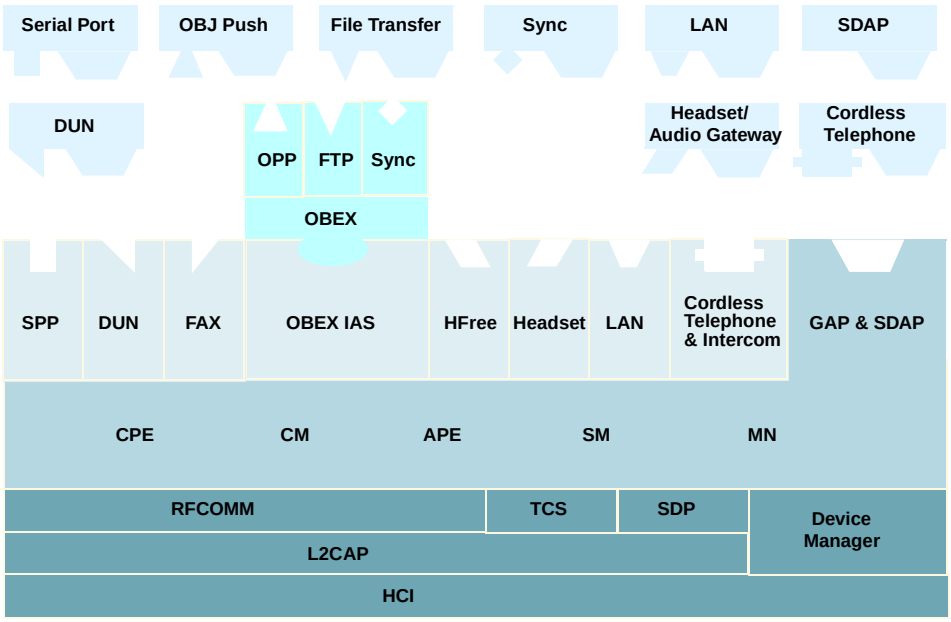
Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/mips_4ke.html

Mezoe Interface Express

Interface Express™ Toolkit including BlueStack® Protocol Stack with Profiles and ProtoDeveloper™ Software

Mezoe's Interface Express Toolkit includes a full Bluetooth™ upper layer stack, OBEX stack, Bluetooth profiles, and source application examples, allowing you to create working Bluetooth applications in just minutes. Mezoe's Bluetooth software connects directly and easily to the DesignWare BlueIQ Core and Bluetooth Development Kit through the standard HCI-over-UART transport. Even designers with little or no Bluetooth experience gain the freedom to quickly create and test their own custom Bluetooth applications on working silicon, when used with the DesignWare Bluetooth Development Kit. These applications may then be implemented in silicon with the DesignWare BlueIQ Core.



During development, designers may use Interface Express on a PC to create applications and immediately test them through an attached DesignWare Bluetooth Development Kit. Mezoe's Bluetooth software, together with the DesignWare Bluetooth Development Kit, allows designers to focus on their application and get their Bluetooth products quickly to market.

Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/mezoe_ip.html

SMSC USB 2.0 PHY

Standard Microsystems Corporation GT3100 USB 2.0 PHY Star IP MacroCell

Synopsys provides simulation and timing models for the Standard Microsystems Corporation (SMSC) GT3100 and GT3200. The GT3100 USB 2.0 PHY Core, created by SMSC, is a USB-IF "Hi-Speed" certified USB 2.0 physical layer (PHY) transceiver available as an intellectual property (IP) core for system-on-a-chip developers of USB 2.0 applications. The GT3200, also from SMSC, is a discrete IC implementation of this PHY core and is also "Hi-Speed" certified.

The SMSC GT3100 and GT3200 incorporate robust clock and data recovery circuitry with integrated termination, providing seamless integration into a wide variety of USB 2.0 applications, as well as backward compatibility with existing USB 1.1 protocol devices. At 165mW absolute maximum power dissipation, the GT3100 operates within the 100mA limit imposed by the USB 2.0 standard. The GT3100 and GT3200 connect directly to a USB SIE via the industry standard 16-bit UTMI interface.

Also see the following web page for additional information:

http://www.synopsys.com/products/designware/starip/smsc_ip.html

SMSC USB 2.0 PHY
Standard Microsystems Corporation GT3100 USB 2.0 PHY Star IP

Index

A

ahb_act_montitor_vmt 258
ahb_bus_vmt 258
ahb_master_vmt 258
ahb_monitor_vmt 258
ahb_slave_vmt 258
AMBA On-Chip Bus IP, listing 17
apb_master_vmt 260
apb_monitor_vmt 260
apb_slave_vmt 260

B

Board and FPGA Verification IP, listing 21
Building Block IP
  application specific - control logic 221
  application specific - interface 218
  data integrity 176
  data integrity - coding overview 184
  datapath - arithmetic overview 33
  datapath - floating point overview 105
  datapath - sequential 98
  datapath - trigonometric overview 94
  datapath generator overview 32
  logic - combinational overview 202
  logic - sequential overview 207
  memory - asynchronous RAMs 163
  memory - FIFO overview 119
  memory - registers 113
  memory - stacks 171
  overview 27
  test - JTAG overview 189

C

Cores, overview 271

D

datapath generator, overview 32
Design Compiler 29
DesignWare Building Block IP, *See also* Building Block IP
DesignWare Core
  dwcore_blueiq_devkit 275
DesignWare Cores, overview 271
DesignWare DSP Library 230
DesignWare FlexModels
  listing 262
DesignWare FlexModels, overview 262
DesignWare Foundation Library, *See also* Building Block IP
DesignWare GTECH Library 230
DesignWare IP Family, overview 15
DesignWare Library Synthesizable IP 27
  AMBA On-chip Bus Logic and Peripherals 231
  Building Block IP 27
  DW_ahb 232
  DW_ahb_h2h 234
  DW_ahb_icm 235
  DW_ahb_ictl 236
  DW_apb 237
  DW_apb_gpio 238
  DW_apb_i2c 239
  DW_apb_ictl 240
  DW_apb_rap 241
  DW_apb_rtc 242
  DW_apb_ssi 243
  DW_apb_timers 245
  DW_apb_uart 246
  DW_apb_wdt 248
  DW_memctl 250
  DW_rambist 251
  DW8051 254
  Memory IP 249
DesignWare Library Verification IP, overview 255
DesignWare Memory Models
  data storage 267
  features 267
  overview 267

DesignWare Star IP, overview 299
DesignWare Synthesizable Core
  dwcore_1394_avlink 290
  dwcore_1394_cphy 293
  dwcore_1394_device 291
  dwcore_1394_ohci 292
  dwcore_blueiq 273
  dwcore_ethernet 276
  dwcore_ethernet_sub 277
  dwcore_gig_ethernet 278
  dwcore_gig_ethernet_sub 279
  dwcore_jpeg_codec 295
  dwcore_jpeg2_codec 296
  dwcore_jpeg2_encod 297
  dwcore_jvxtreme 294
  dwcore_pci 280
  dwcore_pci_express 282
  dwcore_pcix 281
  dwcore_usb_otg 283
  dwcore_usb1_device 284
  dwcore_usb1_host 285
  dwcore_usb1_hub 286
  dwcore_usb2_device 288
  dwcore_usb2_host 287
  dwcore_usb2_phy 289
DesignWare VMT Models, overview 256
DSP Library Overview 230
DW_8b10b_dec 185
DW_8b10b_enc 187
DW_8b10b_unbal 188
DW_add_fp 106
DW_addsub_dx 39
DW_ahb 232
DW_ahb_h2h 234
DW_ahb_icm 235
DW_ahb_ictl 236
DW_apb 237
DW_apb_gpio 238
DW_apb_i2c 239
DW_apb_ictl 240
DW_apb_rap 241
DW_apb_rtc 242
DW_apb_ssi 243
DW_apb_timers 245
DW_apb_uart 246
DW_apb_wdt 248
DW_arbiter_2t 222
DW_arbiter_dp 224
DW_arbiter_fcfs 226
DW_arbiter_sp 228
DW_asymfifo_s1_df 120
DW_asymfifo_s1_sf 123
DW_asymfifo_s2_sf 127
DW_asymfifoctl_s1_df 138
DW_asymfifoctl_s1_sf 141
DW_asymfifoctl_s2_sf 144
DW_bc_1 195
DW_bc_2 196
DW_bc_3 197
DW_bc_4 198
DW_bc_5 199
DW_bc_7 200
DW_bin2gray 42
DW_C166S 304
DW_cmp_dx 48
DW_cmp_fp 107
DW_cntr_gray 50
DW_crc_p 177
DW_crc_s 179
DW_debugger 219
DW_div 53
DW_div_fp 108
DW_div_pipe 55
DW_div_seq 99
DW_dpll_sd 211
DW_ecc 181
DW_fifo_s1_df 131
DW_fifo_s1_sf 133
DW_fifo_s2_sf 135
DW_fifoctl_s1_df 148
DW_fifoctl_s1_sf 150
DW_fifoctl_s2_sf 152
DW_flt2i_fp 110
DW_gray2bin 57
DW_i2flt_fp 111
DW_IBM440 300

DW_inc_gray 61
DW_memctl 250
DW_minmax 63
DW_MIPS4KE 308
DW_mult_dx 72
DW_mult_fp 112
DW_mult_pipe 73
DW_mult_seq 101
DW_prod_sum_pipe 79
DW_ram_2r_w_a_dff 166
DW_ram_2r_w_a_lat 168
DW_ram_2r_w_s_dff 158
DW_ram_2r_w_s_lat 160
DW_ram_r_w_a_dff 164
DW_ram_r_w_a_lat 165
DW_ram_r_w_s_dff 156
DW_ram_r_w_s_lat 157
DW_ram_rw_a_dff 169
DW_ram_rw_a_lat 170
DW_ram_rw_s_dff 161
DW_ram_rw_s_lat 162
DW_rambist 251
DW_shifter 83
DW_sqrt 87
DW_sqrt_pipe 88
DW_sqrt_seq 103
DW_square 85
DW_squarep 86
DW_stack 172
DW_stackctl 174
DW_tap 190
DW_tap_uc 192
DW_TriCore1 306
DW_V850E-Star 302
DW01_absval 34
DW01_add 35
DW01_addsub 37
DW01_ash 41
DW01_binenc 203
DW01_bsh 43
DW01_cmp2 45
DW01_cmp6 46
DW01_csa 51
DW01_dec 52
DW01_decode 204
DW01_inc 58
DW01_incdec 59
DW01_mux_any 205
DW01_prienc 206
DW01_satrnd 81
DW01_sub 90
DW02_cos 95
DW02_mac 62
DW02_mult 64
DW02_mult_2_stage 67
DW02_mult_3_stage 68
DW02_mult_4_stage 69
DW02_mult_5_stage 70
DW02_mult_6_stage 71
DW02_multp 66
DW02_prod_sum 75
DW02_prod_sum1 77
DW02_sin 96
DW02_sincos 97
DW02_sum 92
DW02_tree 93
DW03_bictr_dcnto 208
DW03_bictr_decode 210
DW03_bictr_scnto 209
DW03_lfsr_dcnto 213
DW03_lfsr_load 215
DW03_lfsr_scnto 214
DW03_lfsr_updn 216
DW03_pipe_reg 114
DW03_reg_s_pl 115
DW03_shftreg 116
DW03_updn_ctr 217
DW04_par_gen 183
DW04_shad_reg 117
DW8051 254
dwcore_1394_avlink 290
dwcore_1394_cphy 293
dwcore_1394_device 291
dwcore_1394_ohci 292

dwcore_blueiq 273
dwcore_blueiq_devkit 275
dwcore_ethernet 276
dwcore_ethernet_sub 277
dwcore_gig_ethernet 278
dwcore_gig_ethernet_sub 279
dwcore_jpeg_codec 295
dwcore_jpeg2_codec 296
dwcore_jpeg2_encod 297
dwcore_jvxtreme 294
dwcore_pci 280
dwcore_pci_express 282
dwcore_pcix 281
dwcore_usb_otg 283
dwcore_usb1_device 284
dwcore_usb1_host 285
dwcore_usb1_hub 286
dwcore_usb2_device 288
dwcore_usb2_host 287
dwcore_usb2_phy 289
DWMM, *See also* Memory IP

E

enethub_fx 264
enetrx_fx 264
enettx_fx 264
Ethernet Verification Models 264

F

FlexModels 262
Foundation Library, *See also* Building Block IP
FPGA Compiler II 30
FPGA Verification 21

G

GTECH Library Overview 230

H

Help with products 13

I

Interfaces
  SWIFT, connection for SmartModels 268
IP, Synthesizable
  See also DesignWare Library Synthesizable IP

L

Licensing for Synopsys products 12

M

Memory IP 249
Memory IP, listing 20
Memory Models, *See also* DesignWare Memory Models
Mezoe Interface Express 310
Microprocessors/Microcontroller Cores 253
Microprocessors/Microcontroller Cores, listing 19
Models
  behavioral 268
  FlexModels 262
  SmartModel behavioral simulation 268
  VMT 256
Module Compiler 105

P

PCI / PCI-X Bus Verification Models 265
pcimaster_fx 265
pcimonitor_fx 265
pcislave_fx 265

R

rmiirs_fx 264

S

SCL 12
sio_monitor_vmt 261
sio_txrx_vmt 261

SmartModel Library
  SWIFT interface, connection through 268
SmartModels
  listed in IP Directory Web site 268
SMC USB 2.0 PHY 311
Star IP Core
  DW_C166S 304
  DW_IBM440 300
  DW_MIPS4KE 308
  DW_TriCore1 306
  DW_V850E-Star 302
  Mezoe Interface Express 310
  SMSC USB 2.0 PHY 311
Star IP, overview 299
SWIFT interface
  connection between SmartModels and simulators 268
Synopsys Common Licensing 12
Synthesizable IP, *See also* DesignWare Library Synthesizable IP

U

USB 1.1 / 2.0 Bus Host Interface Model 266
usbhost_fz 266

V

Verification IP, overview 255
Verification Models of Bus and I/O Standards, listing 20
VMT Models, overview 256